RUPRECHT-KARLS-
UNIVERSITÄT HEIDELBERG

3D Oberflächeninterpolation für die interaktive Segmentierung medizinischer Strukturen

Abschlussarbeit zur Erlangung des akademischen Grades
Diplom-Informatiker der Medizin (Dipl.-Inform. Med.)

vorgelegt von
Andreas Fetzer

Referent:	Prof. Dr. Rolf Bendl
Korreferent:	Prof. Dr. Hans-Peter Meinzer
Betreuer:	Dr. Tobias Heimann

Abgabedatum: 15.06.2011

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

(Ort, Datum)	(Unterschrift)

Abstract

Die Segmentierung anatomischer Strukturen ist eine zentrale Aufgabe der medizinischen Bildverarbeitung. Trotz einer Vielzahl vorhandener halb- und vollautomatischer Segmentierungsalgorithmen gilt eine von Experten manuell erstellte Segmentierung auch heute noch als der Goldstandard. Ein großes Problem dabei ist, dass die manuelle Vorgehensweise ein sehr zeitaufwendiger Prozess ist.

In dieser Arbeit wird ein Algorithmus für eine schnelle und interaktive Erstellung einer 3D Segmentierung für das Medical Imaging Interaction Toolkit (MITK) entwickelt. Dafür wird anhand vom Benutzer eingezeichneter Konturen die Oberfläche eines segmentierten Bereiches unter Verwendung radialer Basisfunktionen interpoliert. Die interaktive Komponente wird dabei durch die in MITK bereits existierenden manuellen 2D Segmentierungswerkzeuge gewährleistet.

Um eine gute Beschreibung segmentierter Strukturen anhand weniger Konturen zu ermöglichen, werden die vorhandenen manuellen 2D Segmentierungswerkzeuge so erweitert, dass sie auch in beliebig orientierten Bildebenen einsetzbar sind.

Im ersten Schritt wird die Extraktion von 2D Konturen aus 3D Bildern ermöglicht. Darüber hinaus werden, um die Anzahl der Punkte für die Interpolation möglichst gering zu halten, zwei Algorithmen implementiert, die die Randpunkte der extrahierten Konturen reduzieren.

Aufgrund der Eigenschaft der Interpolation ist es notwendig, die Oberflächennormalen an den Stellen der Konturrandpunkte zu approximieren. Auch hierfür wird ein Algorithmus entwickelt.

Im letzten Schritt wird die gesuchte Oberfläche interpoliert und aus dieser wiederum eine 3D Segmentierung erzeugt.

Die Ergebnisse der Arbeit werden umfassend evaluiert. Es wird die Interpolation an unterschiedlichsten Strukturen mit variierender Anzahl Konturen getestet. Des Weiteren wird die Interpolation für die implementierten Reduzierungsalgorithmen untersucht. Alle erzielten Resultate werden im Anschluss daran kritisch betrachtet und diskutiert.

Danksagung

Bedanken möchte ich mich bei meinen beiden Referenten: Herrn Prof. Dr. Rolf Bendl für die Annahme des Themas und Herrn Prof. Dr. Hans-Peter Meinzer, dass er es mir ermöglicht hat, unter solch angenehmen Arbeitsbedingungen meine Diplomarbeit anfertigen zu können.

Des Weiteren möchte ich der Abteilung MBI des Deutschen Krebsforschungszentrums für das hervorragende Arbeitsklima und die Unterstützung während meiner Diplomarbeit danken. Speziell Herrn Sven Mersmann und Herrn Bastian Graser, die mir vor allem während der Einarbeitungsphase immer mit Rat und Tat zur Seite standen. Vielen Dank auch für das Korrekturlesen meiner Arbeit.

Ganz besonderer Dank geht an meinen Betreuer Herrn Dr. Tobias Heimann: Für die guten Ratschläge, die meine Arbeit immer in die richtige Richtung geleitet haben, für die Bereitstellung guter Datensätze für die Evaluation und vor allem aber für seinen Einsatz beim Korrekturlesen zum Ende der Arbeit hin!

Auch möchte ich mich bei Herrn Prof. Dr. Martin Buhmann für seine fachliche Unterstützung bei Fragen zu den radialen Basisfunktionen bedanken. Ich möchte ausdrücklich betonen, dass mir dadurch sehr weitergeholfen wurde.

Ein dickes Dankeschön geht auch an meine Freundin, die mich gerade in der Schlussphase so sehr unterstützt und an Tagen, an denen nichts voranging, aufgebaut und motiviert hat.

Zu guter Letzt möchte ich meinen Eltern danken! Ohne deren Unterstützung in jeglicher Hinsicht, wäre mir das Studium so sicherlich nicht möglich gewesen. Vielen Dank!

Inhaltsverzeichnis

Abkürzungsverzeichnis

MITK Medical Imaging Interaction Toolkit

VTK Visualisation Toolkit

ITK Insight Segmentation and Registration Toolkit

RBF Radiale Basisfunktion

DP Douglas - Peucker Algorithmus

MPR Multiplanare Rekonstruktion

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Mit der Entdeckung der Röntgenstrahlen im Jahre 1895 begründete Wilhelm Conrad Röntgen die medizinische Bildgebung. Diese ermöglichte erstmals, zu diagnostischen oder therapeutischen Zwecken einen Blick in das Innere des Menschen zu werfen, ohne dass sich dieser einer unter Umständen riskanten Operation unterziehen musste. Die Röntgentechnik gestattet allerdings nur die Projektion anatomischer Strukturen auf ein zweidimensionales Bild.
Erst die Erweiterung der Bildgebung auf tomographische Verfahren, wie der Computer- und Magnetresonanztomographie, erlaubte es, kontrastreiche, überlagerungsfreie 3D Schichtbilder zu erzeugen [13, S. 1].
Weitere Unterstützung hat die medizinische Bildgebung durch die Computertechnik erfahren, die eine digitale Nachbearbeitung der Bilder oder oft auch eine umfassende Bildanalyse ermöglicht. Im Zuge der technischen Weiterentwicklung kommen immer leistungsfähigere Computer zum Einsatz. Daher ist es nicht verwunderlich, dass medizinische Bilder heute hauptsächlich digital gespeichert, verschickt und bearbeitet werden. Auch entwickeln sich die bildgebenden Modalitäten weiter, was zu immer höher aufgelösten Bildern führt, in denen immer feinere Strukturen erkennbar sind. Simultan bedeutet das, dass für immer größere Datenmengen eine digitale Bearbeitung am Computer bewerkstelligt werden muss. Es ist deshalb ein zentrales Anliegen, effiziente bildverarbeitende Algorithmen zu entwickeln.

1.1 Motivation

Ein wichtiges Teilgebiet der medizinischen Bildverarbeitung ist die Segmentierung. Das Ziel einer Segmentierung ist die optische Abgrenzung diagnostisch oder therapeutisch relevanter Bildbereiche, wie Organe, (Tumor-) Gewebe oder auch Gefäße gegenüber dem restlichen Bild. So muss im Rahmen einer Tumorverlaufskontrolle ein Tumor vermessen, dessen Volumen bestimmt oder für eine visuelle Beurteilung ein 3D Modell des Tumors erstellt werden können. Dazu muss dieser in allen Bildbereichen segmentiert werden [13, S. 95]. Eine Segmentierung kann über das manuelle Einzeichnen von Konturen durch medizinische Experten erfolgen. Des Weiteren existieren halbautomatische Segmentierungsalgorithmen, die ein gewisses Maß an Interaktion erfordern, wie das Setzen eines Startpunktes. Auch wurden bereits vollautomatische Algorithmen entwickelt, wie zum Beispiel die 3D Segmentierung mit statistischen Shape Models [14]. Diese benötigen Trainingsdaten als Grundlage, um Formmodelle zu erzeugen. Die Trainingsdaten müssen wiederum von Hand erstellt werden. Ein weiterer Aspekt ist, dass alle halb- oder vollautomatischen Algorithmen bezüglich manuell entstandener Segmentierungen evaluiert werden. Der Grund dafür ist, dass die Qualität medizinischer Bilddaten im klinischen

Alltag häufig durch Rauschen und Artefakte beeinflusst wird, wodurch auch die Ergebnisse der Segmentierungsalgorithmen beeinträchtigt werden. Die von Hand erzeugte Segmentierung wird deshalb generell als Goldstandard angesehen.
Die Segmentierung von Hand ist allerdings für den Mediziner ein sehr zeitaufwendiger Prozess. Es muss dafür in jeder Schicht eines Bildvolumens eine Kontur um den betroffenen Bereich gezeichnet werden. Ist das Bild sehr hoch aufgelöst, oder ist der segmentierte Bereich sehr groß, wie im Falle der Leber, so müssen viele Konturen erstellt werden. Zwischen 100 und 300 Konturen sind dabei keine Seltenheit, was einige Stunden in Anspruch nehmen kann.

1.2 Aufgabenstellung

Das Erstellen einer vollständigen 3D Segmentierung beliebiger Strukturen soll beschleunigt werden. Das Ziel dieser Arbeit ist es deshalb, ein Verfahren für das *Medical Imaging Interaction Toolkit* (MITK) zu implementieren, das aus wenigen Konturen eine möglichst exakte 3D Segmentierung des betrachteten Bereiches erzeugt. Dabei soll einerseits ein hohes Maß an Benutzerinteraktion gewährleistet sein, um den qualitativen Vorzug einer manuellen Segmentierung weitgehend zu erhalten. Andererseits soll der entwickelte Algorithmus einen deutlich reduzierten Zeitaufwand erfordern. Folgende Anforderungen werden an das Verfahren gestellt:

1. Die Konturen, die den segmentierten Bereich beschreiben, sollen manuell vom Anwender eingezeichnet werden können

2. Dafür sollen die bereits vorhandenen manuellen Segmentierungswerkzeuge des Frameworks MITK verwendet werden können

3. Um dem Benutzer möglichst viel Freiheit bei der Beschreibung der segmentierten Struktur zu erlauben, sollen die in MITK vorhandenen Segmentierungswerkzeuge so erweitert werden, dass sie auch in beliebig orientierten Bildebenen einsetzbar sind

4. Die Erstellung der 3D Segmentierung soll anschließend über eine Interpolation der Oberfläche des segmentierten Bereiches auf Basis der eingezeichneten Konturen erfolgen. Die resultierende Oberfläche stellt dabei eine möglichst gute Approximation an die tatsächliche dar

5. Aus der interpolierten Oberfläche soll eine 3D Segmentierung erstellt werden

Da die Rekonstruktion, d.h. die Interpolation von Oberflächen anhand gegebener Punkte ein wichtiges Thema in vielen Gebieten ist, wie zum Beispiel beim *Computer Aided Design (CAD)* oder der Rekonstruktion von Oberflächen aus Laserscans, wird zu Beginn der Arbeit eine umfassende Literaturrecherche durchgeführt, um sich einen Überblick über den aktuellen Stand der Technik zu verschaffen.

2 Grundlagen

2.1 Einführung in das Framework MITK

Wie in Kapitel 1 erwähnt, wird der im Rahmen dieser Arbeit entwickelte Algorithmus für das Framework MITK implementiert. Deshalb widmet sich dieser Teil in erster Linie MITK und den darin vorhandenen Konzepten, welche für die Realisierung dieser Arbeit von Bedeutung sind.

MITK, ITK und VTK

MITK ist keine eigenständige Applikation, sondern ein *Open Source Toolkit*, das die Entwicklung von bildverarbeitender Software unterstützt. MITK basiert zum einen auf dem *Insight Segmentation and Registration Toolkit (ITK)*[1] und erweitert dieses. ITK ist eine Bibliothek, die viele standardisierte Bildverarbeitungsalgorithmen bereitstellt. Die meisten MITK Klassen sind von ITK Klassen abgeleitet, was die Verwendung bereits bewährter Designkonzepte, wie z.B. den Smart-Pointer-, Time-Stamp- und Pipeline-Mechanismus [30] dieser Bibliothek erlaubt.
Zum anderen verwendet MITK für die Visualisierung der verschiedenen Datenobjekte das *Visualization Toolkit (VTK)*[2]. Anders als bei ITK werden MITK Klassen nicht von VTK Klassen abgeleitet, sondern binden diese in den eigenen Rendering Mechanismus ein.
Ein wichtiger Aspekt der medizinischen Bildverarbeitung ist Interaktion. Sie ermöglicht eine (Nach-)Bearbeitung der Bilddaten von Hand und somit einer Verifizierung und Verbesserung der Ergebnisse von bildverarbeitenden Algorithmen. ITK und VTK lassen Benutzerinteraktionen in nur sehr begrenztem Maße zu. Deshalb kombiniert MITK diese beiden Toolkits zusätzlich mit eigenen Konzepten, die eine weitreichende Interaktion erlauben [28].
Ein Konzept, das grundlegend bereits in ITK vorhanden ist, aber von MITK erweitert wurde, ist das Geometrie-Konzept [31]. Da dieses Konzept für die Segmentierung in beliebig orientierten Bildebenen eine entscheidende Rolle spielt, wird es im folgenden Abschnitt ausführlich erklärt.

[1] www.itk.org
[2] www.vtk.org

Das Geometrie-Konzept in MITK

In MITK wird grundsätzlich zwischen zwei verschiedenen Koordinatensystemen unterschieden [31] [8]:

1. Ein Weltkoordinatensystem, in dem alle Datenobjekte (z.B. Bilder oder Oberflächen) liegen.

2. Ein intrinsisches Koordinatensystem für jedes Datenobjekt, das die Lage und Größe des Objektes innerhalb des Weltkoordinatensystems angibt.

Die Informationen, um zwischen intrinsischen und Weltkoordinaten zu konvertieren, werden MITK-seitig in den Geometrie-Klassen gespeichert. Jedes Datenobjekt besitzt ein dabei eigenes Geometrie-Objekt. Das Koordinatensystem eines Bildes wird häufig auch Indexkoordinatensystem genannt, da über dieses auf die diskreten Indizes der Bildpixel zugegriffen werden kann. Im Gegensatz dazu werden Weltkoordinaten in Millimeter angegeben. Abbildung 2.1 zeigt die Beziehung zwischen diesen beiden Koordinatensystemen am Beispiel eines Bildes.

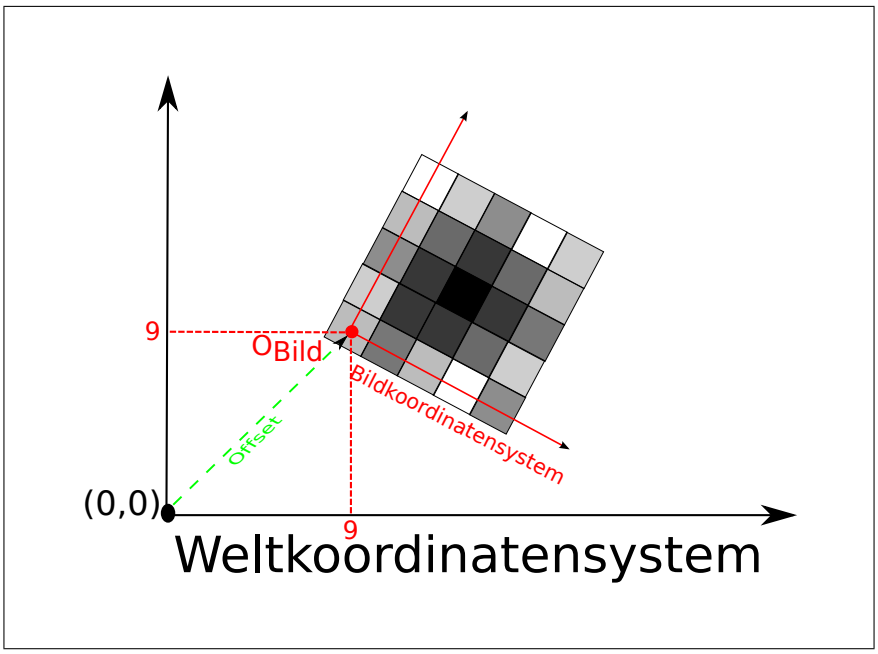


Abbildung 2.1: Die unterschiedlichen Koordinatensysteme in MITK. Datenobjekte, z.B. Bilder, befinden sich aufgrund ihres eigenen Koordinatensystems an einer festen Position im Weltkoordinatensystem

Über die Geometrie eines Bildes lässt sich zwischen Welt- und Indexkoordinaten transformieren. Möchte man zum Beispiel wissen, welchen Pixelwert das Bild an einer bestimmten Position im Weltkoordinatensystem hat, muss man die entsprechenden Weltkoordinaten in Indexkoordinaten transformieren.

Die Transformation wird wie folgt berechnet:

Seien

$$point\_mm = \begin{pmatrix} 53,7 \\ 111,3 \\ 18,9 \end{pmatrix} \quad \text{ein Punkt in Weltkoordinaten,}$$

$$O_{Bild} = \begin{pmatrix} 25,0 \\ 43,0 \\ 6,0 \end{pmatrix} \quad \text{der Ursprung des Bildes in Weltkoordinaten,}$$

$$spacing = \begin{pmatrix} 1,0 \\ 1,0 \\ 3,0 \end{pmatrix} \quad \text{das Pixel-Spacing des Bildes, also die Ausdehnung eines Pixel in allen Dimensionen in Millimeter}$$

Der Ursprung wird im Geometrie-Objekt des Bildes als Offset bzgl. des Ursprungs des Weltkoordinatensystems gespeichert. Sowohl das Pixel-Spacing als auch Informationen über die Rotation des Datenobjektes fließen in die Transformationsmatrix ein. Geht man davon aus, dass die Bildachsen parallel zu den Achsen des Weltkoordinatensystems verlaufen, ergibt sich die folgende Transformationsmatrix:

$$m_{indextoworld} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Für die Umrechnung von Welt- in Indexkoordinaten muss dann lediglich der Abstand des *point_mm* vom Ursprung des Bildes O_{Bild} berechnet werden. Anschließend wird dieser mit der inversen Transformationsmatrix multipliziert:

$$point\_units = (m_{indextoworld})^{-1}[point\_mm - offset]$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{3} \end{pmatrix} \left[\begin{pmatrix} 53,7 \\ 111,3 \\ 18,9 \end{pmatrix} - \begin{pmatrix} 25,0 \\ 43,0 \\ 6,0 \end{pmatrix} \right]$$

$$= \begin{pmatrix} 28,7 \\ 68,3 \\ 4,3 \end{pmatrix} \approx \begin{pmatrix} 29 \\ 68 \\ 4 \end{pmatrix}$$

Analog erfolgt die Rücktransformation von Index- in Weltkoordinaten. Dabei werden, um den Abstand (in mm) vom Bildursprung zu erhalten, die Indexkoordinaten mit der Transformationsmatrix multipliziert. Anschließend wird der Offset wieder dazu addiert:

$$point\_mm = [m_{indextoworld} * point\_units] + offset$$

$$= \left[\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} 28,7 \\ 68,3 \\ 4,3 \end{pmatrix} \right] + \begin{pmatrix} 25,0 \\ 43,0 \\ 6,0 \end{pmatrix}$$

$$= \begin{pmatrix} 28,7 \\ 68,3 \\ 12,3 \end{pmatrix} + \begin{pmatrix} 25,0 \\ 43,0 \\ 6,0 \end{pmatrix} = \begin{pmatrix} 53,7 \\ 111,3 \\ 18,9 \end{pmatrix}$$

Anhand der Geometrie eines Bildes erhalten die Renderer die nötigen Informationen darüber, wie sie das Bild in den jeweiligen Render-Windows anzeigen müssen [31]. Abbildung 2.2 zeigt die für MITK typischen vier Render-Windows. Die 2D Fenster repräsentieren dabei jeweils die Sicht auf ein der anatomischen Körperebenen: transversal (rot), sagittal (grün) und coronal (blau). Das 3D Window stellt die 3D Sicht auf das Bildvolumen dar.

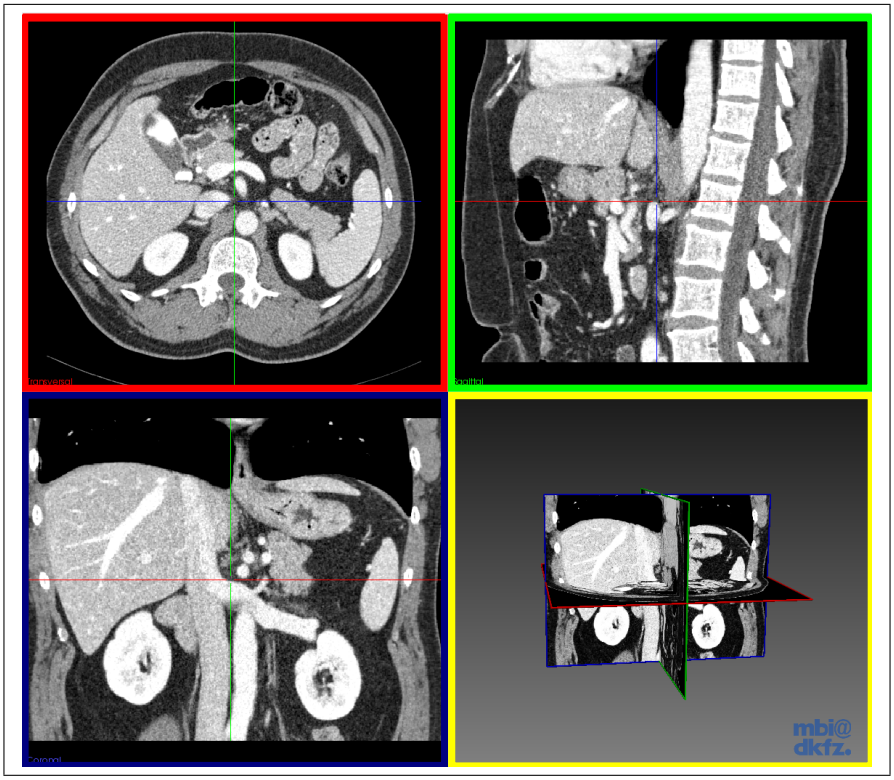


Abbildung 2.2: Die MITK Render-Windows von links oben nach rechts unten: transversale, sagittale, coronale und 3D Ansicht

Da jedes Window eine unterschiedliche Sichtweise repräsentiert, benötigt auch jedes einen eigenen Renderer. Die anzuzeigenden Datenobjekte bzw. deren Geometrien definieren den Bereich über den gerendert werden muss. Folglich hat jeder Renderer selbst auch eine Geometrie. Die Renderer der 2D Windows besitzen zudem zusätzlich eine spezielle 2D Geometrie, die die Position der dargestellten Bildebenen innerhalb des Weltkoordinatensystems beschreibt.
Navigiert man durch die Schichten eines Bildes oder rotiert man eine der Ebenen, so ändert sich deren Position und damit auch die 2D Geometry des zuständigen Renderers.

3 Stand der Technik

In Abschnitt 1.2 wurde beschrieben, dass die Erzeugung einer 3D Segmentierung über die Interpolation der Oberfläche eines segmentierten Bereiches auf Basis der eingezeichneten Konturen erfolgen soll. Daher bildet die Oberflächeninterpolation den Kern dieses Kapitels.

Die Rekonstruktion von Oberflächen ist ein zentrales Thema der Computergrafik und wird für viele Aufgaben benötigt. Ein Anwendungsgebiet findet sich zum Beispiel im Bereich des *Computer Aided Design (CAD)*. Mittels CAD können unter anderem Maschinenteile am Computer modelliert werden. Auf Basis dieser Modelle kann anschließend eine industrielle Fertigung erfolgen.
Oft werden Objekte der realen Welt, wie zum Beispiel Statuen oder auch komplette Terrains, mittels Laserscanner abgetastet. Das Resultat eines solchen Scans ist in der Regel eine große Punktwolke, die am Rechner weiterverarbeitet werden muss.
In Abschnitt 1.1 wurde mit der Tumorverlaufskontrolle ein medizinischer Anwendungsfall genannt. Hier müssen die gesuchten Oberflächen auf irgendeine Weise aus vorhandenen Bilddaten extrahiert werden.
Wie man sieht, werden Oberflächen durch ihre Digitalisierung häufig nur unvollständig beschrieben, wie z.B. durch Punktwolken oder, wie in dieser Arbeit, durch Konturen. Sie müssen deshalb mittels Interpolation rekonstruiert werden.

Es soll nun ein Überblick über die Entwicklung von Verfahren zur Oberflächeninterpolation während der letzten Jahre gegeben werden. Die Ergebnisse der Recherche werden im Folgenden beschrieben, wobei die untersuchten Methoden kritisch in Bezug auf die Aufgabenstellung dieser Arbeit betrachtet werden.

Eine Methode stellten Lee und Lin [17] vor. Sie entwickelten einen Algorithmus zur formbasierten Interpolation. Dazu werden zwischen zwei aufeinanderfolgenden Konturen charakteristische Liniensegmente gesucht, die zueinander in Beziehung gebracht werden (Abbildung 3.1). Anhand der gefundenen Liniensegmente werden anschließend die dazwischenliegenden Schichten interpoliert. Eine Einschränkung bei diesem Verfahren ist, dass es nur für parallele Konturen angewendet werden kann. Um nicht allzu viele Konturen einzeichnen zu müssen, ist es Teil der Aufgabenstellung, eine Interpolation anhand beliebig orientierter Konturen durchzuführen. Der Ansatz von Lee und Lin ist für diese Arbeit deshalb nicht geeignet.

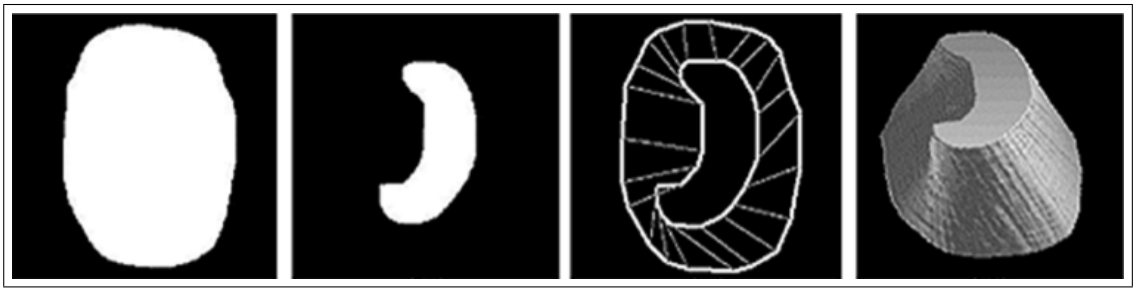
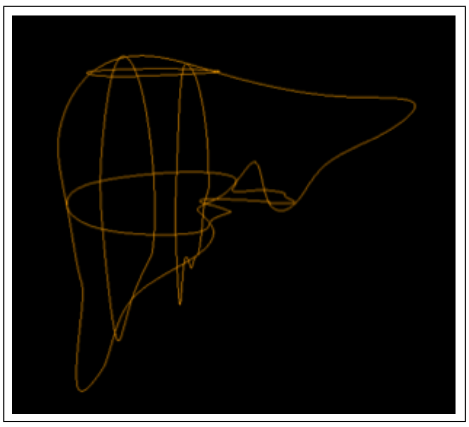
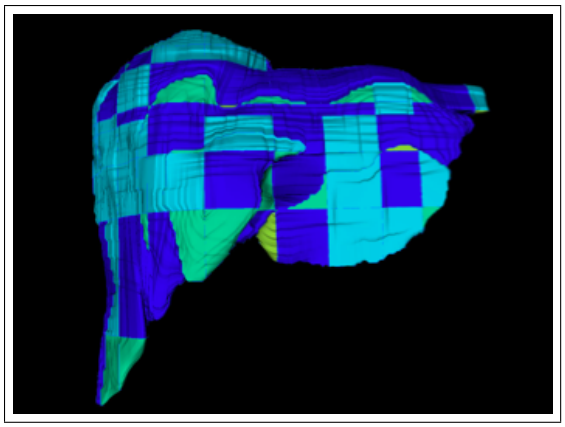


Abbildung 3.1: Das Verfahren von Lee und Lin: Die beiden linken Konturen sind Formen, zwischen denen interpoliert werden soll. Das dritte Bild von links zeigt die Zuordnung charakteristischer Liniensegmente der verschiedenen Konturen zueinander. Im rechten Bild ist das Ergebnis der Interpolation der dazwischenliegenden Schichten zu sehen.

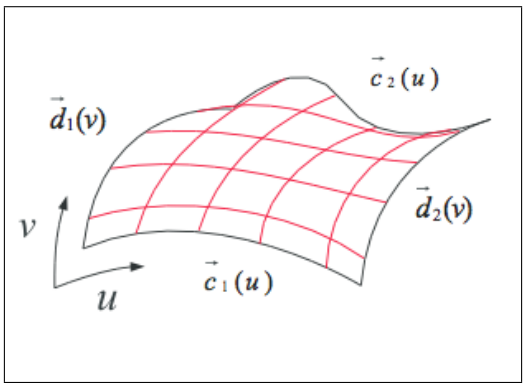
Beliebig orientierte Konturen sind dagegen kein Hindernis für Wolf et. al. [29]. In der dort vorgestellten Methode werden sogenannte *Coons-Patches* zur Oberflächeninterpolation eingesetzt. Dabei wird von den eingezeichneten Konturen ein Netz aufgespannt, das die Oberfläche beschreibt (Abbildung 3.2 (a)). Jede einzelne Netzmasche wird anschließend durch genau einen Coons-Patch bedeckt (Abbildung 3.2 (b)). Dieser wird anhand der Ränder der Netzmasche, also der umliegenden Konturen, berechnet (Abbildung 3.2 (c)). Auch bei diesem Verfahren gibt es Einschränkungen: Ein *Coons-Patch* muss aus mindestens drei, maximal aber vier angrenzenden Konturen entstehen. Für mehr oder weniger Kanten ist das Verfahren nicht ausgelegt. Dies schränkt den Benutzer in seiner Freiheit bei der Beschreibung der Oberfläche ein.
Die eingezeichneten Konturen müssen zudem zueinander konsistent sein. Das bedeutet, dass sich Schnittstellen zwischen den einzelnen Konturen exakt berühren müssen. In der Praxis hat sich herausgestellt, dass dies nicht so ohne Weiteres gewährleistet werden kann, weil die Konturen manuell in die Schichten eingezeichnet werden müssen. Aufgrund dieser Tatsache ist die Coons-Patch Interpolation nicht immer ein stabiles Verfahren.



(a) Die Ränder der eingezeichneten Konturen bilden ein Gitternetz, das die Oberfläche beschreibt.



(b) Jede der so entstandenen Netzmaschen wird anschließend durch einen Coons-Patch bedeckt



(c) Dabei wird jeder Coons-Patch aus den Grenzen der umliegenden Konturen berechnet

Abbildung 3.2: Das Verfahren von Wolf et al.: Die Coons-Patch Interpolation

Eine weitere Möglichkeit ist die Interpolation der Oberfläche mittels *Multi-level Partition of Unity (MPU) Implicit Models*, erstmals vorgestellt von Othake et. al. [20] auch später von Braude et al. verwendet [2]. MPU bedeutet zu Deutsch: *Mehrstufige Zerlegung der Eins*. Die Zerlegung der Eins ist eine Methode, mittels der lokal definierte Approximationsfunktionen zu einer einzigen Globalen zusammengeführt werden können. Die vorhandenen Daten werden bei diesem Ansatz in mehrere Teile unterteilt. Jeder Teilbereich wird dann getrennt approximiert. Die Punkte werden jeweils durch eine implizite Funktion angenähert. Dieser spezielle Funktionstyp wird auch in der hier vorgestellten Arbeit für die Interpolation verwendet, weshalb in Kapitel 4.2, in Abschnitt *3. Implizite Beschreibung*, genauer darauf eingegangen wird. Anschließend wird aus den lokalen Approximationsfunktionen eine Globale erzeugt, indem diese gewichtet zusammengeführt werden. Die summierten Gewichte ergeben dabei die Eins. Abbildung 3.3 und Formel 3.1 und 3.2 verdeutlichen den Vorgang.

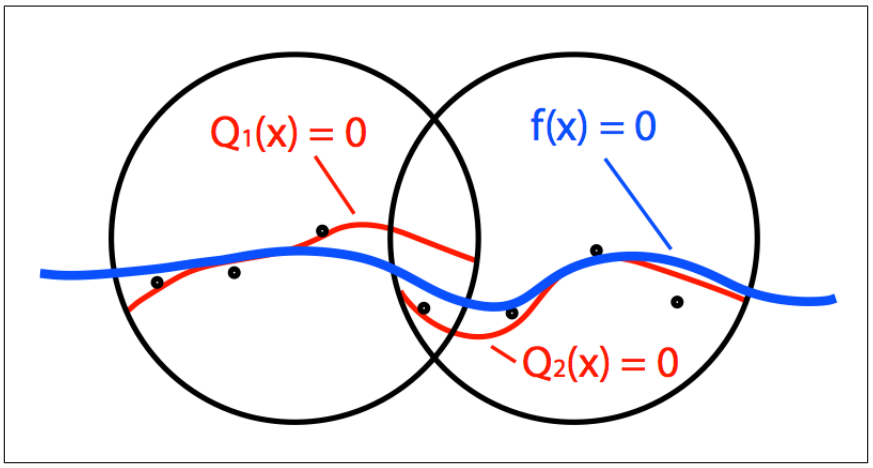


Abbildung 3.3: Erzeugung einer globalen Approximationsfunktion (blau) aus mehreren lokal definierten (rot) [2]

Die globale Funktion $f(x)$ ergibt sich dabei aus:

$$f(x) = \sum_i w_i(x) Q_i(x), \tag{3.1}$$

wobei für die Gewichte w_i gilt:

$$\sum_i w_i = 1 \tag{3.2}$$

Im Falle dieser Arbeit bestehen die vorhandenen Daten aus den Randpunkten weniger Konturen, zwischen denen teilweise große Lücken bestehen können. Eine derartige Datengrundlage ist für eine mehrstufige Unterteilung nicht geeignet.

Die implizite Beschreibung von Oberflächen hat auch in den restlichen betrachteten Ansätzen Verwendung gefunden. In diesen wird die gesuchte implizite Oberflächenfunktion mittels *radialer Basisfunktionen (RBF)* interpoliert. Wie man später in Abschnitt 4.2 sehen wird, wurde die in dieser Arbeit realisierte Interpolation ebenfalls mit RBF durchgeführt, weshalb die theoretischen Hintergründe im genannten Kapitel ausführlicher erläutert werden. Zum besseren Verständnis seien hier einige Fakten vorweggegriffen. Die einfachste Form, eine Approximationsfunktion $f(x)$ mit RBF zu beschreiben ist die folgende [4, S.11-12].

$$f(x) = \sum_{i=1}^{n} \lambda_i \cdot \Phi(||x - x_i||_2)$$

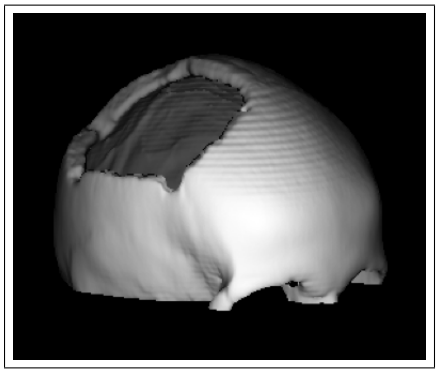
Dabei sind x_i die vorhandenen Punkte für die Interpolation. Wie man sieht, wird die Funktion $f(x)$ als Linearkombination radialer Basisfunktionen $\Phi(x)$ dargestellt. λ_i sind dabei die Gewichte, um die Interpolationsbedingung zu erfüllen. Sie ergeben sich aus der Lösung des folgenden Gleichungssystems für bekannte Punkte x_i und dazu vorhandenen Funktionswerten y_i.

$$\begin{aligned}
\lambda_1 \cdot \Phi(||x_1 - x_1||_2) + \lambda_2 \cdot \Phi(||x_1 - x_2||_2) + \ldots + \lambda_n \cdot \Phi(||x_1 - x_n||_2) &= y_1 \\
\lambda_1 \cdot \Phi(||x_2 - x_1||_2) + \lambda_2 \cdot \Phi(||x_2 - x_2||_2) + \ldots + \lambda_n \cdot \Phi(||x_2 - x_n||_2) &= y_2 \\
&\vdots \\
\lambda_1 \cdot \Phi(||x_n - x_1||_2) + \lambda_2 \cdot \Phi(||x_n - x_2||_2) + \ldots + \lambda_n \cdot \Phi(||x_n - x_n||_2) &= y_n
\end{aligned}$$

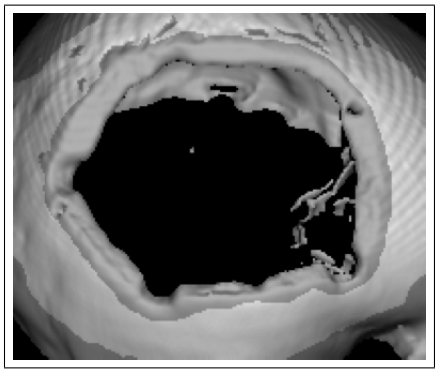
RBF sind gemeinhin dafür bekannt, glatte Approximationsfunktionen für Daten beliebiger Dimensionen zu liefern. Es wird nun ein Überblick über die bereits vorhandenen Lösungen für die Interpolation mit radialen Basisfunktionen und deren Anwendungsgebiete verschafft.
Bereits 1995 benutzten Savchenko et al. [21] RBF zur Interpolation impliziter Funktionen, um Oberflächen mathematisch zu beschreiben.

Eine medizinische Anwendung stellten Carr et al. in [7] vor. Sie reparierten die Oberflächen fehlerhafter Schädel mittels Interpolation, was als Grundlage für die Herstellung von Implantaten für eine Cranioplastie[1] genommen werden kann (Abbildung 3.4).

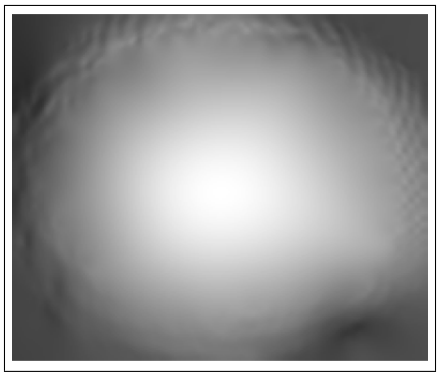
[1] Die Cranioplastie beschäftigt sich mit der Korrektur von Defekten, wie zum Beispiel Löchern in der Schädeldecke.



(a) Rekonstruktion eines defekten Schädels aus CT Daten



(b) Nahansicht der defekten Stelle. Die etwas dunkleren Ränder werden für die Interpolation verwendet



(c) Mittels Interpolation korrigierte defekte Stelle

Abbildung 3.4: Das Verfahren von Carr et al. zur Schädelrekonstruktion

Turk und O'Brien [23] setzten RBF für die Erzeugung von Formtransformationen zwischen zwei unterschiedlichen Formen A und B ein. Sie erzeugten dabei eine Sequenz dazwischenliegender Formen, wobei benachbarte Formen geometrisch ähnlich sein sollten. Dies wurde erreicht, indem man den Koordinaten der Randpunkte der Formen A und B eine zusätzliche Dimension t hinzufügte. Für die eine Form wurde $t = 0$ gesetzt und für die andere $t = t_{max}$. Anschließend wurden alle so entstandenen „Koordinaten" mittels RBF interpoliert und man erhielt genau eine Funktion, die für alle $t \in [0...t_{max}]$ entsprechende Zwischenformen (Abbildung 3.5) lieferte.

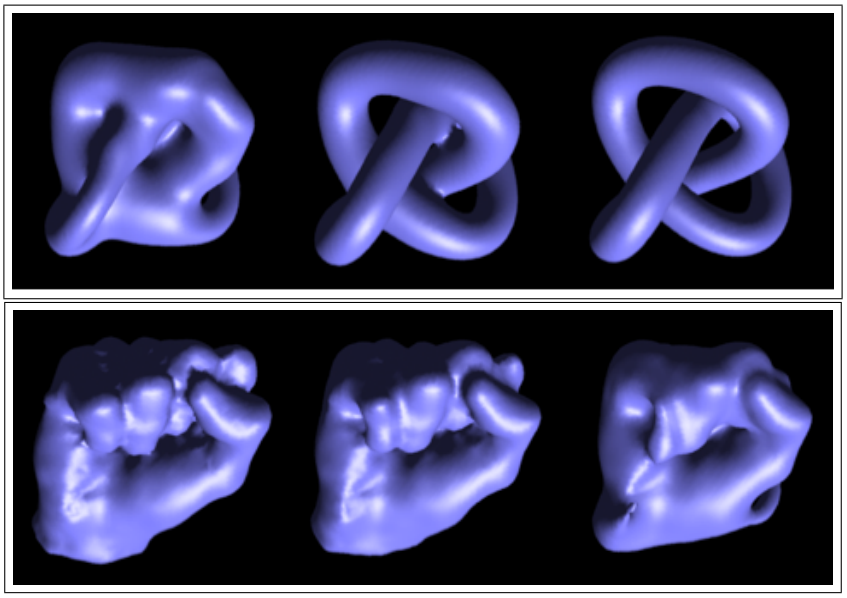


Abbildung 3.5: Das Verfahren von Turk und O'Brien zur Formtransfomation

Die Interpolation mit radialen Basisfunktionen ist ein sehr rechen- und speicherintensives Verfahren, wie man anhand des zu lösenden Gleichungssystems sehen kann. Für N Punkte gilt es im Normalfall N Gleichungen mit N Unbekannten zu lösen. Es ist daher nicht verwunderlich, dass die zuvor vorgestellten Methoden auf kleine Punktmengen begrenzt sind. So würde, wie Carr et al. in [5] schreiben, die Interpolation auf Basis von 544.000 Punkten 4.700 GB an Speicher benötigen, nur um alle erforderlichen Informationen für die Interpolation im Speicher zu halten. In den letzten Jahren wurde deshalb intensiv an Verfahren gearbeitet, um diese Komplexität einzudämmen.

Einer der Vorreiter war Wendland in [24], der erstmals RBF mit kompaktem Träger einsetzte, was dazu führte, dass sich die Menge der für die Interpolation nötigen Informationen stark verringerte. Durch den kompakten Träger haben die RBF bei diesem Ansatz nur einen begrenzten Radius, was bedeutet, dass Änderungen an den Daten lediglich lokale Auswirkungen haben und keine aufwendige Neuberechnungen stattfinden müssen. Auch Morse et al. bauten auf diesem Ansatz auf [18].
Weitere Methoden zur Reduzierung des Berechnungsaufwandes stellen Carr et al. vor [5]. Ihr Ziel war es, die Anzahl der Daten, an welche die Oberflächenfunktion approximiert werden soll, zu reduzieren, ohne dabei merkliche Qualitätseinbußen für das Ergebnis hinnehmen zu müssen.
Hierzu verwenden sie zum einen die *Fast Multipole Method (FMM)*. Eine Methode, bei der Punkte, die sehr nahe beieinander liegen, zu *Clustern* zusammengefasst werden. Diese Cluster werden anschließend als separate Punkte betrachtet, anhand derer die Approximation durchgeführt wird.
Zum anderen wird ein sogenannter *RBF Center Reduction Algorithmus* verwendet, um

die Anzahl der Datenpunkte weiter zu verringern. Dieser nutzt die Tatsache, dass, um ein gutes Ergebnis zu erreichen, nicht alle Punkte in die Interpolation miteinbezogen werden müssen. Carr et al. kombinieren diese Methode mit einem *Greedy Algorithmus*, der zusätzlich ein Fehlermaß für die Approximation berücksichtigt. Somit war es erstmals möglich auch für große Punktmengen eine Interpolation durchzuführen (Abbildung 3.6).

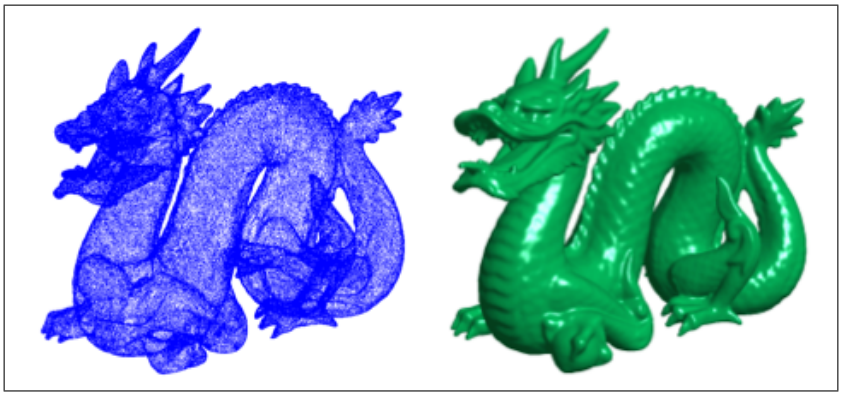
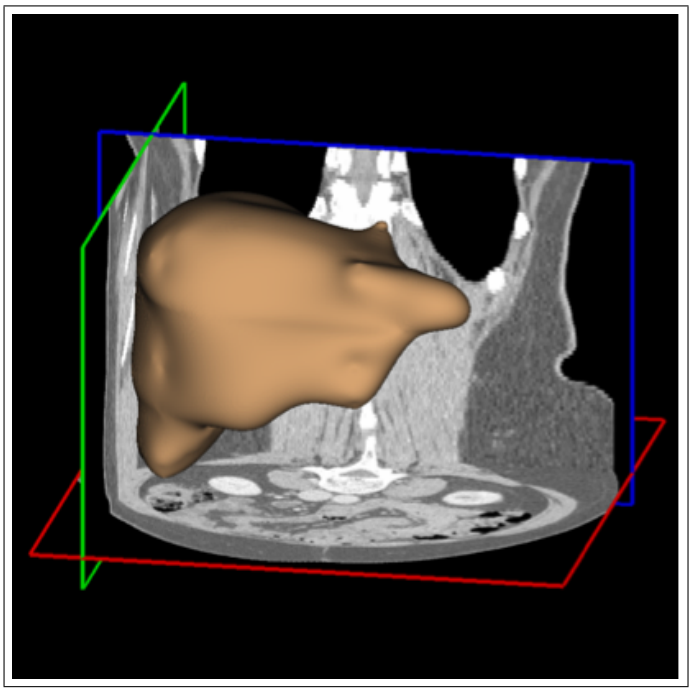


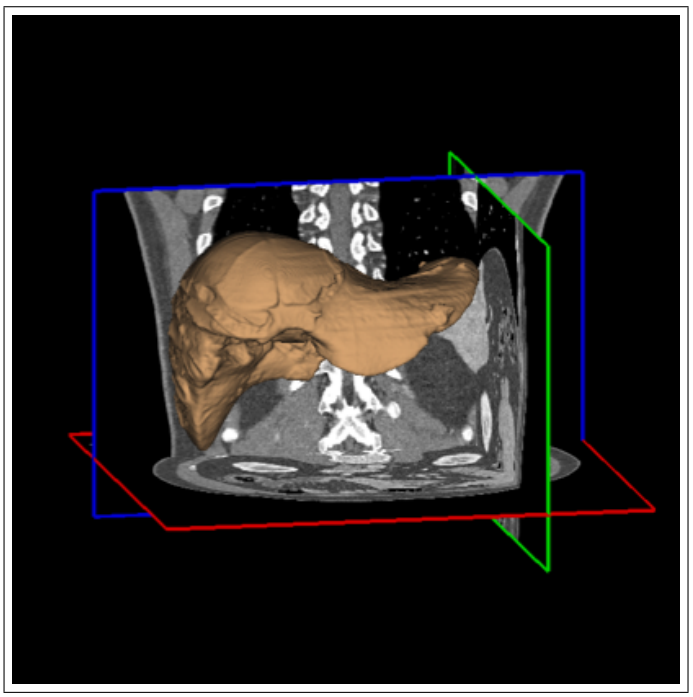
Abbildung 3.6: Die Durchführung der Interpolation mit RBF für große Punktmengen, wie am Beispiel des Drachens für 438.000 Punkte.

Für das Interpolationsproblem in dieser Arbeit liegen allerdings vergleichsweise wenige Punkte vor, da diese aus den Konturrändern extrahiert werden. Die Implementierung aufwendiger Maßnahmen zur Effizienzsteigerung würde deshalb keinen Sinn machen.

Eine ähnliche Problemstellung findet sich dagegen in der Arbeit von Wimmer et al. [27]. Hier wird ein zweistufiges Verfahren zur Organsegmentierung vorgestellt. Der erste Schritt besteht aus dem Platzieren von Kontrollpunkten entlang der Organgrenzen. Durch diese Punkte wird anschließend ein Spline gelegt, der dann an äquidistanten Stellen abgetastet wird. So erhält man die Punkte für die Interpolation. Anhand dieser Punkte wird im nächsten Schritt eine initiale Oberfläche (Abbildung 3.7 (a)) mittels RBF-Interpolation erzeugt. Im letzten Schritt wird die Oberfläche unter Einbeziehung von Bildinformationen anhand der *Level Set Methode* verfeinert (Abbildung 3.7 (b)).



(a) Erzeugung einer initialen Oberfläche



(b) Ergebnis nach der Weiterentwicklung der initialen Oberfläche mittels Level Sets

Abbildung 3.7: Das Verfahren von Wimmer et al.: Die zweistufige Organsegmentierung

4 Material und Methoden

Dieses Kapitel untergliedert sich in zwei Bereiche: die interaktive Segmentierung in beliebig orientierten Bildebenen und die 3D Oberflächeninterpolation. Ersteres bezweckt eine möglichst präzise Beschreibung der gesuchten Oberfläche, ohne dabei in langwieriger Arbeit viele Konturen in das Bildvolumen einzeichnen zu müssen.
Im zweiten Abschnitt wird mit Hilfe dieser Konturen eine Oberfläche berechnet, anhand der eine 3D Segmentierung erstellt wird. Je genauer die Oberfläche beschrieben wurde, desto genauer ist letztendlich auch die Segmentierung.

4.1 Interaktive Segmentierung in beliebig orientierten Ebenen

In diesem Abschnitt wird einleitend die interaktive Segmentierung des Frameworks MITK beschrieben. Im Anschluss daran wird dargestellt, wie diese erweitert wurde, um auch in beliebig orientierten Bildebenen zu funktionieren.

4.1.1 Einleitung

Im Kapitel 1 wurde erklärt, dass man unter einer Segmentierung das optische Hervorheben zusammengehörender Bildbereiche versteht. Die Kennzeichnung solcher Bereiche kann voll- oder halbautomatisch, aber auch komplett manuell erfolgen. Das Framework MITK stellt dem Benutzer eine Reihe von manuellen Segmentierungswerkzeugen zur Verfügung. Mit diesen lassen sich in den anatomischen Körperebenen (Abbildung 4.1) Konturen in das Bildvolumen zeichnen.

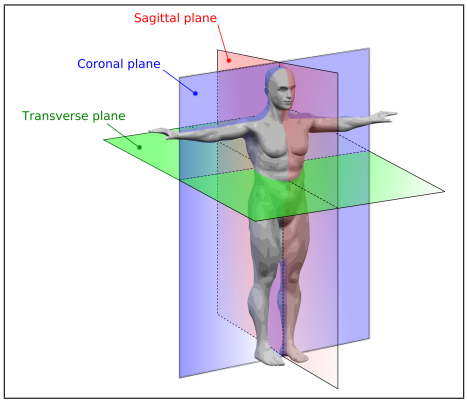


Abbildung 4.1: Die anatomischen Körperebenen [19]

Die Generierung beliebiger 2D Schnittbilder aus originären 3D Bilddaten wird Multiplanare Rekonstruktion (MPR) genannt [13, S.300]. In Abbildung 4.2 kann man eine coronale MPR für eine Computertomographie des Abdomens sehen. Darin lässt sich beispielsweise manuell die Kontur der Leber in die gewählte Schicht einzeichnen:

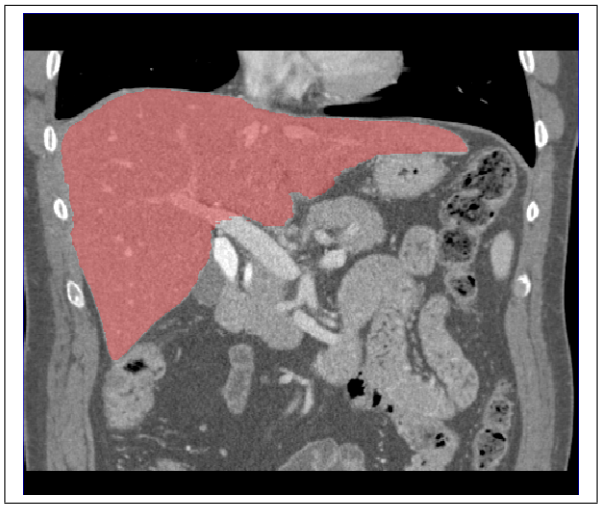


Abbildung 4.2: Eine Leberkontur in der coronalen Ansicht einer MPR des Abdomens

Der programminterne Ablauf (Abbildung 4.3) ist dabei relativ einfach. Zunächst wird eine leere Segmentierung (*Working Image*) mit denselben geometrischen Eigenschaften wie denen des Originalbildes (*Reference Image*) erstellt. Eine Segmentierung ist in diesem Fall ein Binärbild, d.h. ein Bild dessen Pixel lediglich die Werte Null oder Eins annehmen können. Wird vom Benutzer mit Hilfe eines der manuellen Segmentierungswerkzeuge (*Active Tool*) eine Kontur in eine der drei Ebenen eingezeichnet, so wird die entsprechende Schicht aus dem Bildvolumen extrahiert. Die eingezeichnete Kontur wird in die extrahierte Schicht gefüllt und diese wird anschließend zurück in das Bildvolumen geschrieben. Die von der Kontur eingeschlossenen Pixel haben dann auch im Originalbild den Wert eins.

Ein Ziel dieser Arbeit ist eine aussagekräftige Beschreibung der Oberfläche der segmentierten Struktur mit möglichst wenigen Konturen. Um dies zu ermöglichen, ist es sinnvoll, nicht nur in den Standardebenen Konturen einzuzeichnen, sondern auch in Ebenen, die schräg durch das Bildvolumen verlaufen.
Die Form einer Kugel beispielsweise lässt sich mit den drei Standardebenen einfach darstellen. Möchte man aber die Form von Objekten beschreiben, die schräg in einem Bild liegen, wie zum Beispiel Knochen oder Gefäße, dann müssten in den Standardebenen unnötig viele Konturen eingezeichnet werden, um das betreffende Objekt ausreichend gut zu charakterisieren. Könnte man aber diese Ebenen beliebig drehen und in diese rotierten Positionen Konturen einzeichnen, so würde sich die Anzahl der Arbeitsschritte deutlich reduzieren.

In MITK ist es möglich, die Ebenen zur Ansicht zu rotieren. Allerdings war es nicht möglich, in diese gedrehten Schichten auch Konturen zu zeichnen. Es sollen nun kurz die für die Schichtextraktion vorhandenen Werkzeuge in MITK betrachtet werden.

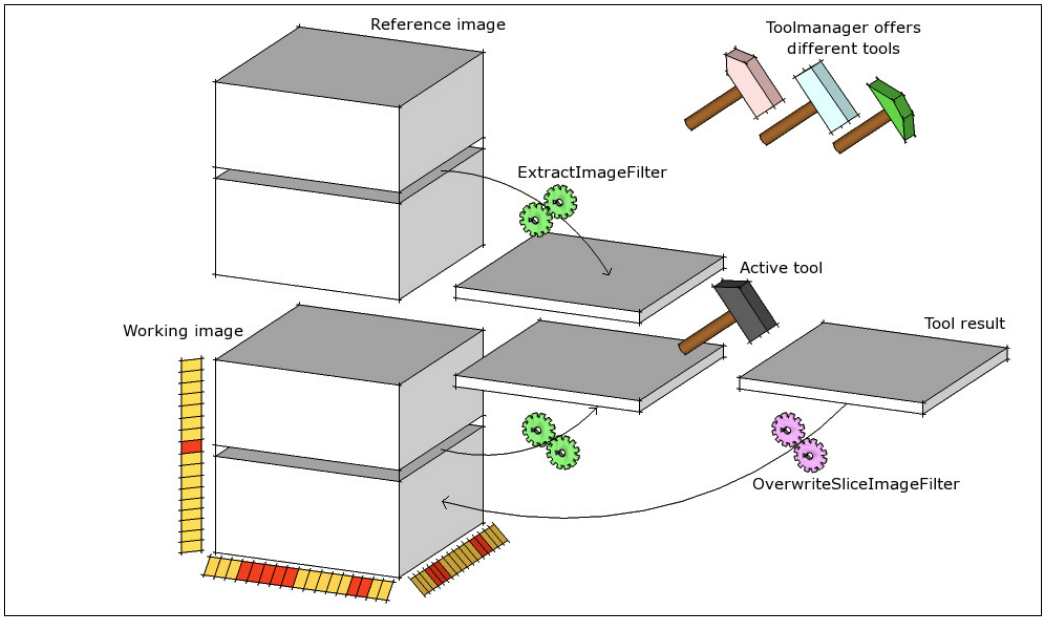


Abbildung 4.3: Ablauf einer manuellen Segmentierung [9]

Der mitkExtractImageFilter und mitkOverwriteSliceImageFilter

In Abbildung 4.3 kann man sehen, dass zum Extrahieren eines gewählten Schichtbildes der `mitkExtractImageFilter` verwendet wird. Dieser liest anhand der Schichtnummer und der Orientierung (transversal, sagittal oder coronal) das entsprechende Schichtbild aus dem Bildvolumen aus.
Nachdem das Schichtbild durch das *Active Tool* bearbeitet wurde, wird es vom `mitkOverwriteSliceImageFilter` wieder zurück in das Ursprungsbild geschrieben.
Auch dieser muss dafür die Position des Bildes und die entsprechende Orientierung im 3D Bild kennen. Orientierung und Schichtnummer reichen allerdings nicht aus, um Rotationen abzubilden. Es müssen also neue Filterklassen implementiert werden, die die entsprechenden Informationen erhalten, um beliebig orientierte Schichten aus einem Volumen herauslesen zu können.

4.1.2 Implementierung

MITK gestattet dem Benutzer ein hohes Maß an Interaktion. Interagiert ein Benutzer in einem der 2D Render-Windows (z.B. durch das Einzeichnen einer Kontur in eine Schicht), so wird von dem zuständigen Renderer ein `PositionEvent` versendet. Die Segmentierungswerkzeuge können ein solches Event abfangen und verarbeiten. Über dieses kann auf dessen Sender, also den Renderer, zugegriffen werden. In Abschnitt 2.1 wurde beschrieben, dass eben genau dieser Renderer die Informationen über die Lage der aktuellen Ebene im Raum in Form einer 2D Geometrie kennt. Diese 2D Geometrie wird innerhalb des Renderes in der Membervariable `m_CurrentWorldGeometry2D` bereitgestellt. Um nun in beliebig orientierten Schichten segmentieren zu können, muss sowohl eine Klasse, die anhand einer solchen Geometrie eine Schicht aus einem 3D Volumen extrahieren kann implementiert werden, als auch eine Klasse, die selbiges wieder zurück schreiben kann.
Es gibt zwar in MITK den `mitkExtractDirectedPlaneImageFilter`, der mit Hilfe des `vtkImageReslice` eine beliebig orientierte Schicht aus einem Bild extrahiert, aber dennoch hat man sich für eine Neu-Implementierung entschieden. Der Grund hierfür ist, dass in dem Bildvolumen, aus dem die Schichten extrahiert werden sollen, bereits Informationen vorhanden sein können. In vorhergegangenen Arbeitsschritten eingezeichnete Konturen können die Schicht, die herausgelesen werden soll, schneiden. Solche Schnittkonturen werden sowohl mit extrahiert, als auch anschließend wieder zurückgeschrieben. Es muss also garantiert sein, dass beide Filterklassen exakt auf den gleichen Pixeln operieren, um einerseits eine korrekte Projektion der eingezeichneten Kontur in das Bildvolumen zu gewährleisten, andererseits um sicherzustellen, dass der bereits vorhandene Bildinhalt durch den Vorgang nicht verändert wird.

ExtractDirectedPlaneImageFilterNew

Damit die Erzeugung einer Segmentierung auch in beliebig orientierten Bildebenen möglich ist, wurde in dieser Arbeit der `mitkExtractDirectedPlaneImageFilterNew` geschrieben. Die Idee dabei ist, das Originalbild entlang der rotierten Ebene abzutasten, also ein sogenanntes Resampling durchzuführen. Dieser Filter benötigt dazu die folgenden Informationen:

1. Das 3D Bild, aus dem die Schicht herausgelesen werden soll
2. Die Geometrie des 3D Bildes
3. Die `m_CurrentWorldGeometry2D` des Renderers

Die Extraktion der Schicht geschieht dann in den folgenden vier Schritten, die anschließend ausführlicher erläutert werden:

1. Erzeugung eines 2D Bildes, welches das Abtastgitter und damit auch die zu extrahierende Schicht repräsentiert

2. Transformation der Indexkoordinaten des erzeugten 2D Bildes mittels der `m_CurrentWorldGeometry2D` in Weltkoordinaten
3. Transformation der erhaltenen Weltkoordinaten in die entsprechenden Indexkoordinaten des 3D Bildes über dessen Geometrie
4. Füllen der entsprechenden Pixelwerte des 3D Bildes in das 2D Schichtbild

In Schritt 1 wird das Schichtbild erzeugt, das aus dem 3D Volumen herausgelesen werden soll. Um problemlos über dessen Bildpunkte iterieren zu können, wird es als ITK Bild erzeugt. Bei der Erzeugung eines ITK Bildes müssen das Pixel-Spacing und die Pixelanzahl in jeder Dimension angegeben werden [16, S.35-37]. Bei der Wahl dieser Parameter muss einiges beachtet werden, denn die Koordinatenachsen der `m_CurrentWorldGeometry2D` des Renderers beschreiben zwar die Rotation, allerdings wird weder das Spacing, noch die Größe der Ebene verändert.
Abbildung 4.4 zeigt ein Bildvolumen durch das eine schräge Ebene verläuft. Rot markiert sind mögliche Pixel, die die Ebene berührt. Es ist deutlich erkennbar, dass die Entscheidung, ob ein Pixel ausgelesen wird oder nicht, von der Lage der schwarzen Abtastpunkte abhängt.

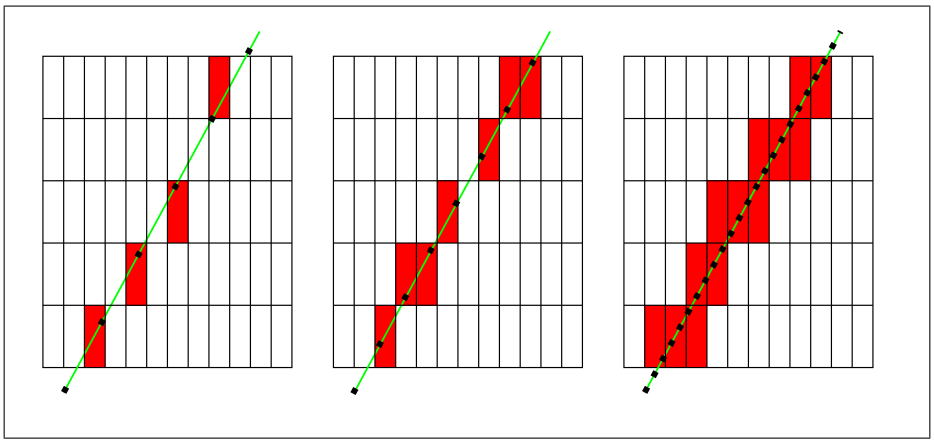


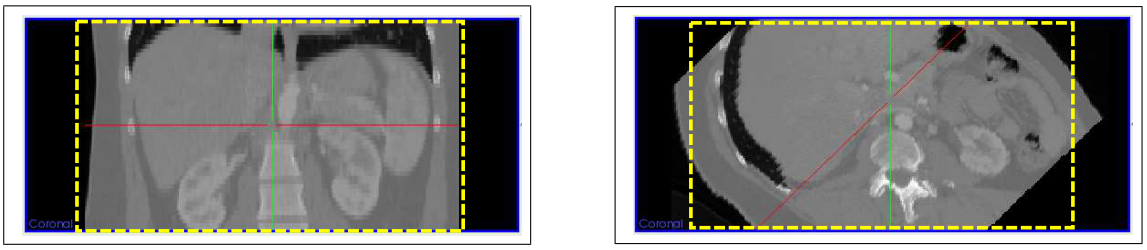
Abbildung 4.4: Abtastproblematik

Da das Spacing des erzeugten Bildes die Schrittweite des Abtastgitters definiert, ist es auch offensichtlich, dass je kleiner das Spacing der rotierten Ebene ist, desto mehr Pixel im Originalbild berührt werden und desto genauer auch die Abtastung erfolgt. Um also eine möglichst korrekte und lückenlose Anzeige der eingezeichneten Segmentierung zu erreichen, sollte darauf geachtet werden, dass das Spacing der schrägen Ebenen klein genug ist, um alle relevanten Pixel des 3D Bildes miteinzubeziehen. Daher wurde als Spacing das halbe kleinste, im Originalbild vorkommende, Spacing festgesetzt.

Zusätzlich zum Pixel-Spacing muss die Größe, d.h. die Anzahl der Pixel, für das Bild gesetzt werden. Damit der komplette Bildbereich von der rotierten Ebene abgedeckt wird,

muss für diese eine ausreichende Größe gewählt werden.

Schließlich gilt es noch eine Besonderheit zu berücksichtigen: Wie bereits erwähnt, beschreibt die `m_CurrentWorldGeometry2D` den maximalen Bereich, der gerendert wird. Allerdings wird der maximale Bereich durch die `DisplayGeometry`, die jeder Renderer hat, begrenzt. Die `DisplayGeometry` hängt von der Größe des Render-Windows ab und stellt eine rechteckige Sicht auf das Weltkoordinatensystem dar. Abbildung 4.5 (b) zeigt ein 2D Render-Window, dessen Ebene um zwei Achsen rotiert wurde. Wie man erkennen kann, liegen die Ecken des Bildes außerhalb des gelben Rechteckes, welches von der `DisplayGeometry` vorgegebenen ist. Sie werden deshalb bei der Segmentierung nicht berücksichtigt.



(a) Nicht rotierte Ebene (b) Rotierte Ebene

Abbildung 4.5: Begrenzung der Renderer Geometry durch die Display-Geometry (gelbes Rechteck)

Gelöst wurde dieses Problem, indem die Größe der `m_CurrentWorldGeometry2D` neu gesetzt wurde. Eine Änderung der Größe der zu extrahierenden Schicht wirkt sich jedoch nur auf den oberen und rechten Bildbereich aus. Daher muss deren Geometrie, genauer gesagt deren Ursprung, zusätzlich nach unten verschoben werden, um auch den unteren und linken Bildbereich erreichen zu können.
Konkret realisiert wurden diese Anpassungen, indem man zuerst eine zu der `m_CurrentWorldGeometry2D` identische Geometrie erzeugt. Dieser wurde dann das neue Pixel-Spacing gesetzt und anschließend wurde die Länge der Bilddiagonalen berechnet. Empirisch hat sich herausgestellt, dass die folgenden Berechnungen der Größe und der Position des Ursprungs zu einer vollständigen Einbeziehung aller Bildbereiche führt. Die maximale Ausdehnung, die eine Ebene aufgrund einer Rotation in einem Render-Window annehmen kann, ist so groß wie die Diagonale der Ebene:

$$\begin{aligned} \texttt{maxExtent} &= \sqrt{\texttt{xExtent}^2 + \texttt{yExtent}^2} \\ \texttt{xTranslation} &= \texttt{maxExtent} - \texttt{xExtent} \\ \texttt{yTranslation} &= \texttt{maxExtent} - \texttt{yExtent} \end{aligned}$$

$$\texttt{size}[0] = \frac{\texttt{maxExtent} + \texttt{xTranslation}}{\texttt{spacing}[0]}$$

$$\texttt{size}[1] = \frac{\texttt{maxExtent} + \texttt{yTranslation}}{\texttt{spacing}[1]}$$

Zu guter Letzt wird der Ursprung der Geometrie verschoben:

$$\begin{aligned}
&\texttt{Point3D right} &&= \texttt{m_CurrentWorldGeometry2D}->\texttt{GetAxisVector}(0)\\
&\texttt{Point3D bottom} &&= \texttt{m_CurrentWorldGeometry2D}->\texttt{GetAxisVector}(1)\\
&\texttt{Point3D origin} &&= \texttt{m_CurrentWorldGeometry2D}->\texttt{GetOrigin}()\\
&\\
&\texttt{origin} &&= \texttt{origin} - (\texttt{xTranslation} * \texttt{right} + \texttt{yTranslation} * \texttt{bottom})
\end{aligned}$$

Mit der so erhaltenen Geometrie werden alle Indexkoordinaten des 2D Bildes in Schritt 2 in Weltkoordinaten transformiert. Weiter in Schritt 3 werden diese Weltkoordinaten über die Geometrie des 3D Bildes wiederum in Indexkoordinaten umgewandelt, allerdings nun bezogen auf das Originalbild. Da die modifizierte `m_CurrentWorldGeometry2D` zum Teil auch außerhalb der Bildvolumens liegt, muss lediglich überprüft werden, ob die aktuell berechneten Indexkoordinaten innerhalb des Bildes liegen. Trifft dies zu, wird der entsprechende Pixelwert in Schritt 4 für das 2D Bild gesetzt. Abbildung 4.6 veranschaulicht die einzelnen Schritte.
Abschließend eine Anmerkung: Bei einer MPR werden in der Regel die neu entstandenen Bildpunkte aus den Benachbarten interpoliert. Das ist in diesem Fall nicht erwünscht, da es sich bei dem Bild, aus dem die Schicht extrahiert wird, um eine Segmentierung und damit ein Binärbild handelt. Wie eingangs erwähnt, können und sollen deshalb die einzelnen Pixel nur die Werte Null oder Eins annehmen, was bei einer Interpolation nicht gewährleistet wäre.

OverwriteDirectedPlaneImageFilter

Analog zur Extraktion einer rotierten Ebene läuft deren Zurückschreiben ab. Um dies zu ermöglichen, wurde der `mitkOverwriteDirectedPlaneImageFilter` geschrieben. Dieser Filter benötigt die folgenden Informationen:

1. Das 3D Bild, in das die Schicht zurückgeschrieben werden soll
2. Die Geometrie des 3D Bildes
3. Die Geometrie des rotierten 2D Bildes

Das Zurückschreiben läuft in drei Schritten ab:

1. Transformation der 2D Indizes in Weltkoordinaten
2. Transformation der Weltkoordinaten in Indexkoordinaten des 3D Bildes
3. Setzen des korrespondierenden Pixelwertes des Schichtbildes an der Stelle der aktuellen Indexkoordinaten im 3D Bild

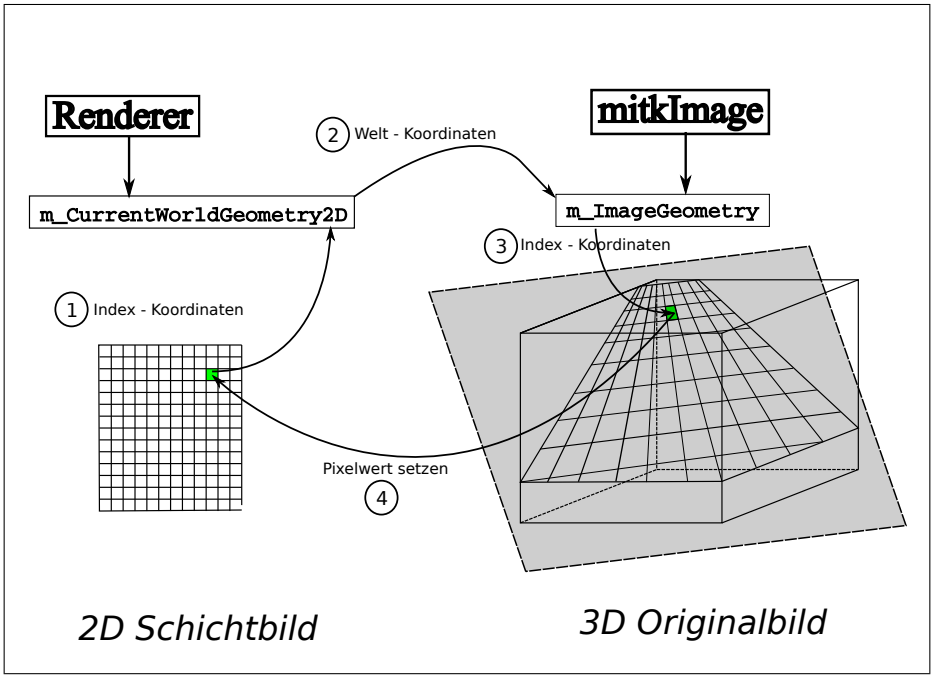


Abbildung 4.6: Schema der Extraktion einer rotierten Ebene

Navigation zwischen rotierten Schichten

Die unterschiedlichen Ebenen müssen in MITK immer manuell rotiert werden. Gerade bei Rotationen um zwei verschiedene Achsen wird es sehr schwierig, eine bestimmte Position aus einem früheren Arbeitsschritt exakt wiederherzustellen. Das bedeutet, dass eine Nachbearbeitung bereits eingezeichneter Konturen fast unmöglich ist. Bereits kleine Unterschiede in der Rotation können dazu führen, dass nur noch ein Teil der ursprünglich eingezeichneten Pixel erreicht wird. Wird dann z.B. eine Kontur gelöscht, kann es vorkommen, dass einige Pixel nicht berücksichtigt werden und im Originalbild erhalten bleiben. Allein vom Standpunkt der Benutzbarkeit aus wäre es deshalb sinnvoll, eine Möglichkeit zu haben, um zwischen mehreren rotierten Schichten navigieren zu können. Die Positionen der Konturen müssen also in irgendeiner Form gespeichert werden. Dazu

werden die Geometrien, die zur Extraktion verwendet werden, als Knoten im Datamanager von MITK gespeichert. Klickt der Anwender auf einen solchen Knoten, so wird die entsprechende Ebene zu der entsprechenden Position rotiert.

4.2 3D Oberflächen-Interpolation

4.2.1 Einleitung

Das Ziel dieser Arbeit ist eine schnelle und qualitativ gute 3D Segmentierung medizinischer Strukturen. Dazu soll die Oberfläche eines segmentierten Bildbereiches anhand weniger Konturen interpoliert werden. In den folgenden Abschnitten werden die mathematischen Grundlagen beschrieben, die benötigt werden, um aus zweidimensionalen Konturen 3D Oberflächen zu berechnen.

Mathematische Beschreibung von Oberflächen

Um aus zweidimensionalen Konturen eine dreidimensionale Oberfläche interpolieren zu können, benötigt man zuallererst eine mathematische Beschreibung der gesuchten Oberfläche. Theoretisch können Oberflächen als geschlossene Kurven mit einer festen Position und Ausrichtung innerhalb eines Koordinatensystems betrachtet werden. Für die Beschreibung geschlossener Kurven gibt es nach Bloomenthal [1] verschiedene Möglichkeiten:

1. Explizite Beschreibung
2. Parametrische Beschreibung
3. Implizite Beschreibung

Diese drei Darstellungsmethoden werden nun am Beispiel einer Kugeloberfläche veranschaulicht.

1. Explizite Beschreibung

Eine Kugel mit dem Mittelpunkt im Ursprung und dem Radius 1 kann anhand der folgenden Gleichung beschrieben werden:

$$x^2 + y^2 + z^2 = 1$$

Gesucht ist also eine mathematische Darstellung, die diese Bedingung erfüllt. Explizite Funktionen haben die allgemeine Form (hier in 2D):

$$f(x, y) = z$$

Die z-Koordinate wird in Abhängigkeit von x und y ausgedrückt. Versucht man die oben genannte Bedingung auf diese Form abzubilden, ist leicht erkennbar, dass sich die gesuchte Oberfläche nicht mit nur einer Funktion darstellen lässt. Denn:

$$\begin{aligned} z^2 &= 1 - x^2 - y^2 \\ z &= \pm\sqrt{1 - x^2 - y^2} \end{aligned}$$

Damit erhält man für die explizite Beschreibung der Kugeloberfläche folgende zwei Funktionen:

$$z = f_1(x, y) = +\sqrt{1 - x^2 - y^2}$$
$$z = f_2(x, y) = -\sqrt{1 - x^2 - y^2}$$

Abbildung 4.7 zeigt die Kugeloberfläche, die man erhält, wenn die Funktionen $f_1(x, y)$ (im Bild grün) und $f_2(x, y)$ (im Bild dunkelblau) zusammen geplottet werden.

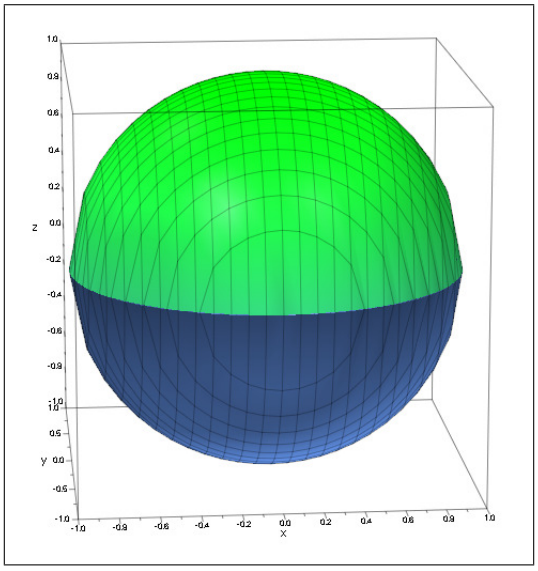


Abbildung 4.7: Explizite Darstellung der Oberfläche einer Kugel. Die zwei dafür benötigten Funktionen beschreiben jeweils die beiden Hemisphären (grün und dunkelblau)

Der Vorteil von expliziten Funktionen ist, dass mit ihnen die gesuchte Oberfläche sehr schnell berechnet und gezeichnet werden kann, da man durch einsetzen von x und y die z-Koordinate erhält und damit explizit jeden Punkt auf der Oberfläche.
Der Nachteil allerdings ist, wie man an dem betrachteten Beispiel gesehen hat, dass eine einzige Funktion nicht ausreichend ist, um die komplette Oberfläche zu beschreiben. Mit expliziten Funktionen können weder vertikale, noch sich selbst überlappende oder schneidende Kurven dargestellt werden. Deshalb werden je nach gesuchter Kurve mindestens zwei Funktionen benötigt [1].
Die explizite Beschreibung bringt noch eine weitere Schwierigkeit mit sich. Gerade in dieser Arbeit ist die Oberflächenfunktion gänzlich unbekannt und muss durch eine Interpolation approximiert werden. Für „primitive“ Oberflächen, wie die der Kugel, lässt sich deren explizite Funktion relativ leicht annähern. Hat man es aber mit komplexeren Oberflächen zu tun, z.B. der Oberfläche der Leber, so ist es äußerst aufwendig bzw. nahezu unmöglich, eine explizite Beschreibung zu finden.

2. Parametrische Beschreibung

Parametrisch beschriebene Oberflächen haben die Form [1],

$$F(s,t) = \begin{pmatrix} f_x(s,t) \\ f_y(s,t) \\ f_z(s,t) \end{pmatrix}$$

Jeder Punkt der Oberfläche wird in Abhängigkeit der Parameter (s,t) ausgedrückt. Am Beispiel der Kugel sieht die parametrische Darstellung folgendermaßen aus:

$$F(s,t) = \begin{pmatrix} rcos(s)sin(t) \\ rsin(s)sin(t) \\ rcos(t) \end{pmatrix}, mit \quad s = 0..2\pi \quad und \quad t = 0..\pi$$

Wie die explizite Beschreibung liefert die parametrische unmittelbar die Punkte auf der Kugeloberfläche. Auch bei dieser Form der Oberflächenbeschreibung ist man sehr eingeschränkt, sogar noch mehr als bei expliziten Funktionen. Eine Parameterdarstellung lässt sich nur für bestimmte, sehr einfache Topologien angeben, wie zum Beispiel für Kreise, Ellipsen, Kugeln, Geraden oder Ebenen. Die Parametrisierung komplexer Formen, wie sie im Körper vorkommen, ist zwar möglich, aber nur mit Hilfe komplexer Funktionen.

3. Implizite Beschreibung

Eine weitere Herangehensweise ist die Beschreibung von Oberflächen mittels impliziter Funktionen. Bloomenthal beschreibt in [1] implizite Oberflächen als einen schmalen Bereich einer messbaren Größe, wie z.B. Farbe, Temperatur oder anderer physikalischer Größen. Der Wert dieser Größe variiert innerhalb des betrachteten Bildbereiches, ist aber direkt auf der Oberfläche konstant. Man kann deshalb sagen, dass die implizite Funktion für alle Oberflächenpunkte den gleichen konstanten Wert liefert. Mathematisch wird eine solche Anforderung als Funktion f ausgedrückt, deren Funktionsargument ein dreidimensionaler Punkt p ist:

$$f(p) = f(x,y,z) = c$$

Man hat hier im Vergleich zu den beiden vorigen Funktionsarten eine zusätzliche Eingabedimension. Dabei ist c ein Punkt im R^n und f erfüllt folgende Abbildung: $R^3 \mapsto R^n$. In den meisten Anwendungen ist $n = 1$ und c damit eine skalare Größe. Die Funktion f selbst kann auf dreierlei Arten spezifiziert werden:

1. Mathematische Funktionen
2. Prozedurale Methoden
3. Diskrete Abtastwerte

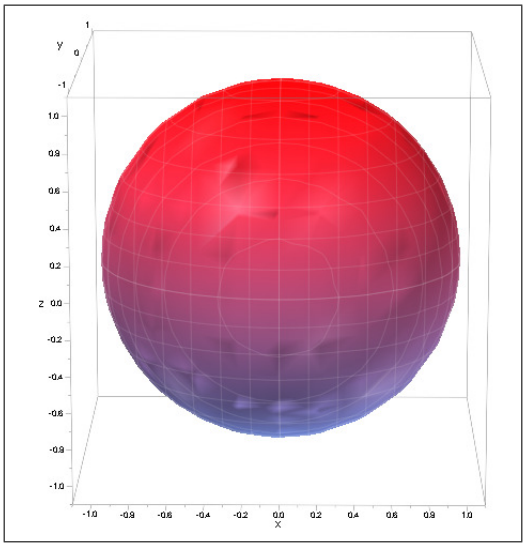


Abbildung 4.8: Implizit beschriebene Kugeloberfläche. Für die Darstellung ist hier eine einzige Funktion ausreichend

In dieser Arbeit wird f ausschließlich durch diskrete Abtastwerte spezifiziert, weshalb die anderen Möglichkeiten im Folgenden außer Acht gelassen werden.
Wie bereits erwähnt, bilden Konturen die Grundlage für die Durchführung der Interpolation der Oberfläche. Genauer gesagt werden lediglich die Randpunkte der Konturen betrachtet, welche als eine Art „diskreter Abtastwerte“ die implizite Funktion mit der letztlich die Oberfläche beschrieben werden soll, definieren.
Im Gegensatz zu den beiden vorigen mathematischen Beschreibungsformen lässt sich mit einer impliziten Funktion kein konkreter Punkt auf der Oberfläche berechnen, sondern zu einem gegebenen Punkt bestimmen, ob dieser innerhalb ($f < c$), außerhalb ($f > c$) oder genau auf der Oberfläche ($f = c$) liegt.
Die implizite Darstellung der Kugeloberfläche ist:

$$f(p) = f(x, y, z) = x^2 + y^2 + z^2 = 1$$

Oder anders ausgedrückt:

$$f(p) = f(x, y, z) = x^2 + y^2 + z^2 - 1 = 0$$

Abbildung 4.8 zeigt das Ergebnis, wenn die implizite Kugelfunktion geplottet wird. Ein großer Vorteil dieser Darstellungsform ist, dass man, vorausgesetzt man kennt zumindestens ein paar Punkte der Oberfläche, für beliebige Topologien eine entsprechende implizite Funktion angeben kann.
Im nächsten Abschnitt wird eine sehr häufig benutzte implizite Funktion beschrieben, mit der im weiteren Verlauf gearbeitet wird.

Die Distanzfunktion

In der Praxis werden implizite Funktionen bereits seit langem zur Oberflächeninterpolation eingesetzt, wie z.B. in den Arbeiten von Savchenko et al. [21], Turk und O'Brien [23] oder Carr et al. [5].
Alle Ansätze haben dabei eine Sache gemeinsam: Sie benutzen für die Interpolation eine besondere implizite Funktion, genannt die *Distanzfunktion*. Auch sie hat die allgemeine Form:

$$f(x, y, z) = c$$

c ist hier ein Maß für den Abstand eines Punktes p zur gesuchten Oberfläche. Ein Punkt liegt genau dann auf der Oberfläche, wenn er den Abstand 0 als Funktionswert ergibt:

$$\{p \in R^3 : f(p) = c = 0\}$$

Des Weiteren bekommen Punkte, die innerhalb des betrachteten Objektes liegen, eine Distanz < 0 und Punkte, die außerhalb liegen eine Distanz > 0 zugewiesen. Abbildung 4.1 zeigt den Verlauf der Distanzfunktion.

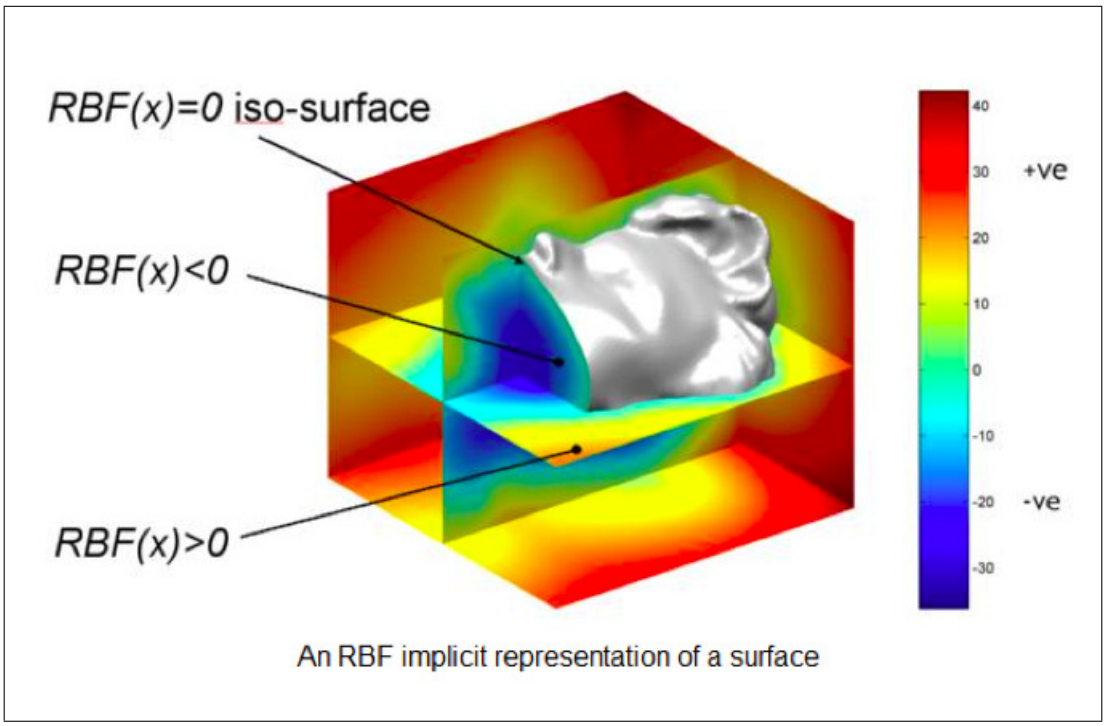


Abbildung 4.9: Der Verlauf der Distanzfunktion [12]. Sie hat an allen Stellen, an denen die Oberfläche verläuft den Wert 0, an Stellen innerhalb der Oberfläche einen Wert < 0 und außerhalb einen Wert > 0

Auch für die Oberfläche der Kugel lässt sich eine Distanzfunktion angeben. Diese sieht folgendermaßen aus:

$$d(p) = d(x, y, z) = x^2 + y^2 + z^2 - 1 = \begin{cases} < 0, & \text{wenn } p \text{ innerhalb der Oberfläche liegt} \\ = 0, & \text{wenn } p \text{ genau auf der Oberfläche liegt} \\ > 0, & \text{wenn } p \text{ außerhalb der Oberfläche liegt} \end{cases}$$

Die Randpunkte der eingezeichneten Konturen liegen per Definition auf der gesuchten Oberfläche und spezifizieren damit die Distanzfunktion. Sie erhalten die Distanz 0 als Funktionswert zugewiesen. Mit der Distanzfunktion hat man nun die Möglichkeit, die gesuchte Oberfläche durch genau eine Funktion mathematisch zu beschreiben. Mit ihr lässt sich für jeden Bildpunkt ein entsprechender Distanzwert berechnen. Die Oberfläche verläuft genau durch die Bildpunkte deren Distanzwert gleich Null ist. Die Pixel innerhalb des Volumens haben folglich negative Distanzwerte und die außerhalb positive. Der nächste Schritt ist nun die Durchführung der Interpolation auf Basis der vorhandenen Punkte und Funktionswerte.

Die Interpolation der Oberfläche

Nachdem alle Konturen eingezeichnet wurden, gilt es die folgende Problemstellung zu lösen:

Für die Menge aller Konturrandpunkte (und somit auch Oberflächenpunkte) X,

$$X = \{x_i\}_{i=1}^{N},\ mit\ x_i \in R^3$$

und den Funktionswerten F einer unbekannten (Oberflächen-)Funktion $f : R^3 \mapsto R$,

$$F = \{f_i\}_{i=1}^{N},\ mit\ f_i = f(x_i) \in R$$

wird eine Approximationsfunktion $d : R^3 \mapsto R$ gesucht, für die gilt:

$$d(x_i) = f(x_i) = f_i$$

Die Approximationsfunktion $d(x)$ beschreibt dabei implizit die gesuchte Oberfläche in Form der Distanzfunktion. Sie wird durch Interpolation auf Basis der gegebenen Punkte x_i berechnet. Mathematisch existieren unterschiedliche Interpolationsmethoden. Für

diese Arbeit wurde auf die Interpolation mit radialen Basisfunktionen zurückgegriffen. Diese sind, wie Wendland [25] schreibt, bereits seit langer Zeit eine beliebte Wahl für die Interpolation. Wendland führt dies auf die folgenden Tatsachen zurück:

1. RBF können in beliebigen Dimensionen verwendet werden
2. Sie funktionieren auch mit verstreuten Daten, die nicht anhand eines bestimmten Gitters verteilt sind
3. Sie erlauben Interpolationsfunktionen von beliebiger „Glattheit“
4. Sie haben eine einfache Struktur, was ihre Verwendung vereinfacht

Auch Buhmann betont in [4, S.1 - 2], dass sich zur Interpolation solcher Problemstellungen RBF besonders gut eignen. Im nächsten Abschnitt folgt die Definition der RBF.

Definition

Carr et al. definieren radiale Basisfunktionen mit der folgenden Notation, die auch in dieser Arbeit verwendet wird [5]:

$$d(x) \quad = \quad p(x) + \sum_{i=1}^{n} \lambda_i \cdot \Phi(||x - x_i||_2) \text{ mit}$$

Φ	ist eine eindimensionale stetige Funktion
λ_i	sind reellwertige Gewichte für die Interpolation
x_i	sind die Zentren der RBF
$\|\|.\|\|_2$	ist die euklidische Distanz
$p(x)$	ist ein Polynom niedrigen Grades

Eine Funktion ist radial bzw. radialsymmetrisch, wenn deren Funktionswert nur von der euklidischen Distanz des Argumentes zum Ursprung abhängt. Im Falle der radialen Basisfunktionen wird das Argument zusätzlich um die Punkte x_i, die auch Zentren der RBF genannt werden, verschoben. Diese Zentren werden später an den vorhandenen Konturrandpunkten platziert. Die Approximationsfunktion $d(x)$, hier die Distanzfunktion, wird also als Linearkombination von Verschiebungen einer eindimensionalen stetigen Funktion $\Phi(x)$, der sog. radialen Basisfunktion, dargestellt[4, S.3]. Das Polynom $p(x)$ wird in bestimmten Fällen benötigt, um zu gewährleisten, dass das Interpolationsproblem immer eindeutig lösbar ist [3]. Eine ausführlichere Erläuterung findet sich im nächsten Abschnitt.

Für die Distanzfunktion bedeutet das, dass sie der folgenden Bedingung gerecht werden muss:

$$d(x) = p(x) + \sum_{i=1}^{n} \lambda_i \cdot \Phi(||x - x_i||_2) = f_x \tag{4.1}$$

Unter Einbeziehung aller gegebenen Konturrandpunkte gilt es das hieraus resultierende Gleichungssystem zu lösen:

$$\begin{pmatrix} \Phi(||x_1 - x_1||_2) & \Phi(||x_1 - x_2||_2) & \dots & \Phi(||x_1 - x_n||_2) \\ \Phi(||x_2 - x_1||_2) & \Phi(||x_2 - x_2||_2) & \dots & \Phi(||x_2 - x_n||_2) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi(||x_n - x_1||_2) & \Phi(||x_n - x_2||_2) & \dots & \Phi(||x_n - x_n||_2) \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \tag{4.2}$$

kurz:

$$A \cdot \lambda = F \tag{4.3}$$

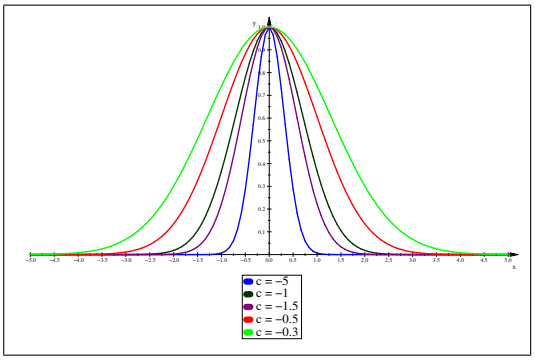
Die Lösungsmatrix A und damit auch $\Phi(x)$ haben dabei besondere Eigenschaften, wie der nächste Abschnitt zeigt.

Die besonderen Eigenschaften der RBF für die Oberflächen - Interpolation

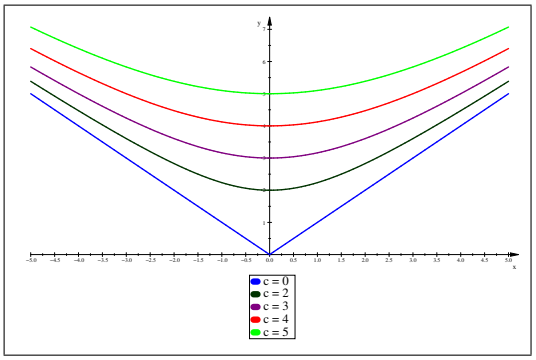
In der Literatur wird oft darauf hingewiesen, dass sich radiale Basisfunktionen besonders gut für die Interpolation von mehrdimensionalen Interpolationsproblemen eignen, wie bereits auf Seite 32 dargestellt. Im Folgenden werden nun die zwei wichtigsten Eigenschaften der RBF betrachtet, die zu dieser Eignung führen.
Zunächst muss erwähnt werden, dass es unterschiedliche Typen radialer Basisfunktionen gibt. Tabelle 4.1 zeigt die am häufigsten verwendeten [5], Abbildung 4.10 deren Kurvenverläufe. Da das Funktionsargument von der euklidischen Distanz abhängt kann es nicht negativ werden, was auch an den Kurvenverläufen ersichtlich ist.

Typ	Form	Verwendung
Thin-Plate Spline	$\Phi(x) = x^2 log(\|x\|)$	für die Interpolation von 2D Funktionen
Gaussian	$\Phi(x) = e^{-cx^2}$	hauptsächlich für neuronale Netze
Multiquadrics	$\Phi(x) = \sqrt{x^2 + c^2}$	eher bei topografischen Daten
Biharmonic	$\Phi(x) = \|x\|$	für die Interpolation von 3D Funktionen
Triharmonic	$\Phi(x) = \|x\|^3$	für die Interpolation von 3D Funktionen

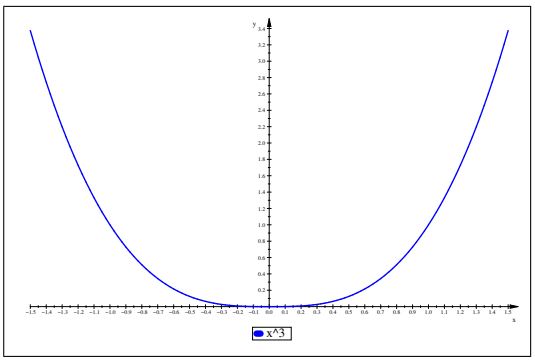
Tabelle 4.1: RBF-Typen



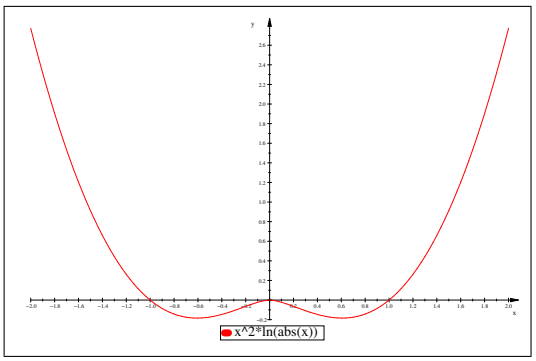
(a) Gaussian RBF



(b) Mulitquadrics und als deren Spezialfall Biharmonic RBF ($c = 0$)



(c) Triharmonic RBF



(d) Thin-Plate-Spline

Abbildung 4.10: Die unterschiedlichen Typen der radialen Basisfunktionen

In [4, S.3] erwähnt Buhmann, dass Interpolationsprobleme für mehrdimensionale und unregelmäßig im Raum verteilte Daten schnell zu singulären Problemen führen können. Häufig trifft man bei Spline- oder Polynominterpolation auf solche Schwierigkeiten. Buhmann beweist im weiteren Verlauf, dass die Lösungsmatrix A aus Gleichung 4.3 im Falle einer Interpolation mit RBF immer invertierbar ist, und damit eine eindeutige Lösung für das Gleichungssystem existiert. In manchen Fällen ist A sogar positiv definit, zum Beispiel bei Verwendung der Gaussian RBF. Der Grund dafür ist, dass die Gaußfunktion eine vollständig monotone Funktion ist, für welche Schoenberg gezeigt hat, dass die Matrix A immer positiv definit ist [4, S. 13]. Für manch andere Typen der radialen Basisfunktionen ist A dagegen nur bedingt positiv definit, wie zum Beispiel für den Thin-Plate-Spline. Um eine Lösbarkeit und in diesem Sinne eine Invertierbarkeit der Matrix A für alle Daten zu gewährleisten, müssen solche Typen durch ein zusätzliches Polynom und eine darüber formulierte Nebenbedingung unterstützt werden [3, S. 3]. Für die multiquadric und die biharmonic RBF kann das Polynom dagegen vernachlässigt werden [3, S.3], [4, S.14-15].
Ein zusätzlicher Vorteil bei der Verwendung von RBF ist laut Buhmann, dass der Funktionswert von der euklidischen Distanz abhängt. Infolgedessen ist dieser invariant ge-

genüber Rotationen. Außerdem wird dadurch aus jedem mehrdimensionalen Problem ein eindimensionales, was eine Berechnung erleichtert [4, S.3-4].

Eine weitere besondere Eigenschaft der radialen Basisfunktionen ist, dass sie sehr glatte Interpolationsfunktionen liefern. Carr et al. [5] begründen diese Tatsache damit, dass radiale Basisfunktionen eine kleine Halbnorm besitzen, was bedeutet, dass sie stetig und unendlich oft differenzierbar sind. Ihre Kurven haben daher keine Ecken, was bezogen auf die Interpolation zu guten Ergebnissen führt. Noch präziser sind Carr et al. in [6], wo sie die biharmonic und triharmonic RBF als die glattesten RBF-Typen bezeichnen. Diese Tatsache ließ sich auch empirisch bei der Interpolation eines Kreises bestätigen (Abbildung 4.11). Man kann erkennen, dass die Gaußfunktion oft zwischen den Stützstellen ausbricht, wohingegen die anderen Funktionen annähernd gleichwertige Ergebnisse liefern. Es fällt auf, dass die multiquadrische RBF im Gegensatz zu den anderen RBF-Typen an manchen Stellen nicht durch die gegebenen Punkte verläuft, was aber auch von dem Wert für c anhängt (siehe Tabelle 4.1). Die Resultate der biharmonischen und triharmonischen RBF sind fast identisch. Ein wichtiger Faktor ist aber auch deren Berechnungsaufwand, der für die Biharmonische deutlich kleiner ist, da diese nur aus der euklidischen Distanz besteht. Aufgrund dieser Erkenntnisse wurde für die Interpolation der Oberfläche in dieser Arbeit die biharmonische radiale Basisfunktion gewählt.

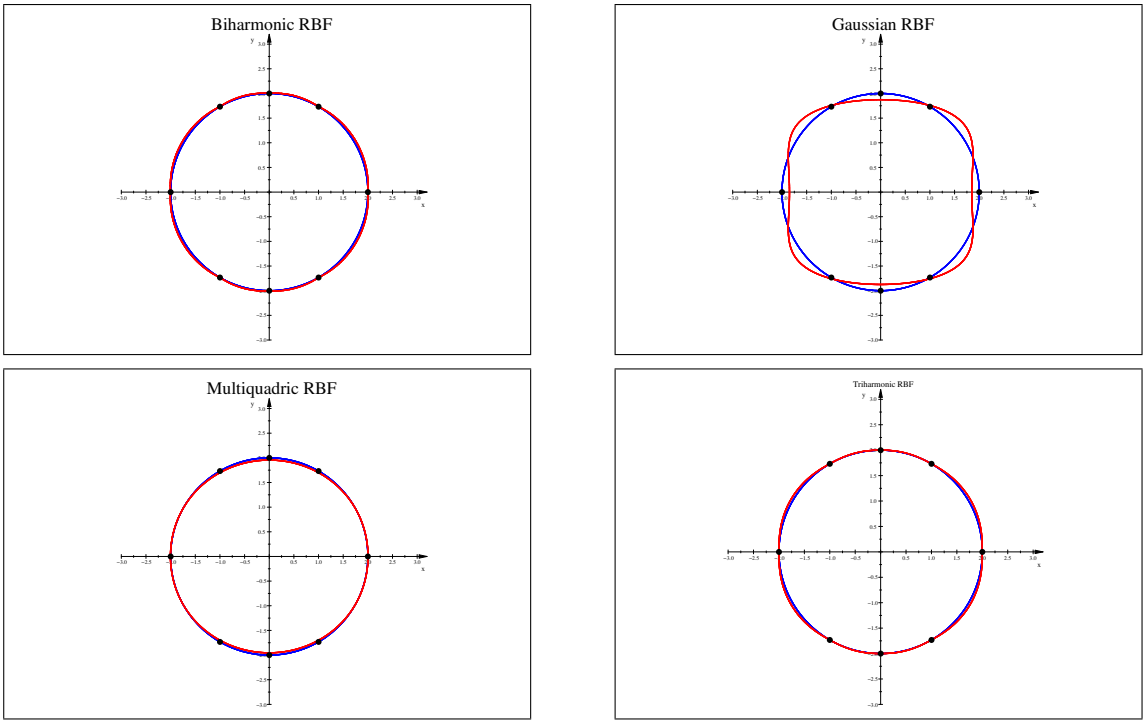


Abbildung 4.11: Die Interpolation eines Kreises (blau) mit unterschiedlichen RBF-Typen. Die Zentren der RBF sind als schwarze Punkte eingezeichnet und der interpolierte Kreis ist rot dargestellt.

Bedeutung der Oberflächennormalen für die Berechnung

Die Gleichungen 4.2 und 4.3 beschreiben, wie erwähnt, das zu lösende Gleichungssystem für die Interpolation. Da alle Punkte x_i auf der gesuchten Oberfläche liegen und somit per Definition den Distanzwert 0 zugewiesen bekommen, ist der Vektor F gleich dem Nullvektor. Die Lösung eines Gleichungssystem, das unter jeder Bedingung 0 als Ergebnis liefert, stellt die triviale Lösung 0 dar.
Um das zu vermeiden, müssen zusätzliche Punkte gefunden werden, die eine Distanz ungleich Null bzgl. der gesuchten Oberfläche haben, sog. *Offsurface-Points*. Diese neuen Distanzwerte müssen mit in den Lösungsvektor F geschrieben werden. In der Praxis wird dieses Problem meist dadurch gelöst, dass an jedem Randpunkt einer Kontur die Normale der Oberfläche approximiert wird. Addiert man den normierten Normalenvektor auf den entsprechenden Punkt, so erhält man einen Punkt, der außerhalb des Zielvolumens liegt. Aufgrund der Norm hat er die Distanz 1. Umgekehrt, subtrahiert man den Normalenvektor, erhält man einen „inneren“ Punkt, der die Distanz -1 hat.
Wimmer et al. [27] benutzen dafür die 2D Normalen an den Konturrandpunkten und erweitern diese an den Schnittpunkten zweier Konturen auf 3D. Dabei muss beachtet werden, dass die äußeren Punkte nicht wieder innerhalb der Oberfläche liegen. Carr et al. beschreiben die Auswirkungen davon in [6], die in Abbildung 4.12 veranschaulicht wird.

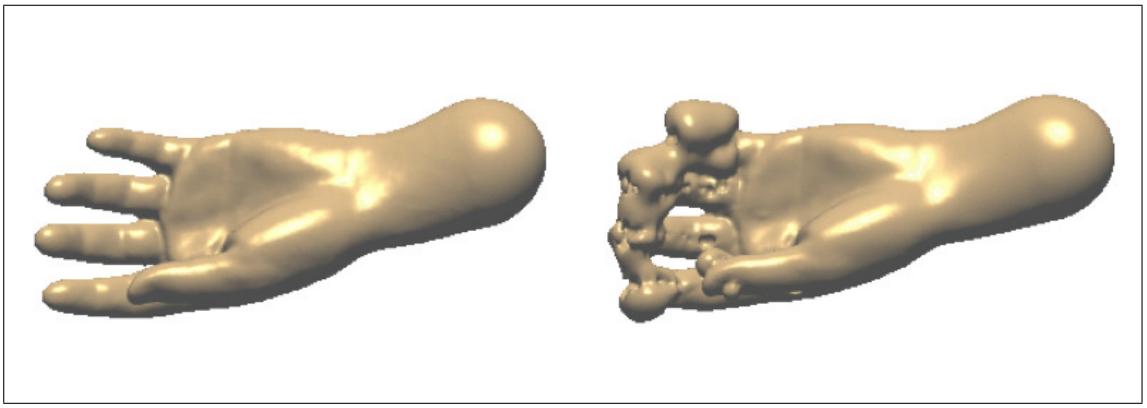


Abbildung 4.12: Bei der rechten Oberfläche liegen die „äußeren“ Punkte aufgrund zu langer Normalen wieder innerhalb der Volumens, was zu einer falschen Interpolation führt [6]

Berechnungsaufwand

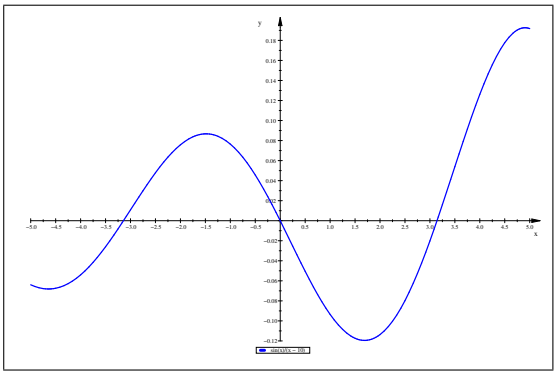
Der Berechnungsaufwand des Interpolationsproblems lässt sich in drei Bereiche unterteilen [18]:

1. Konstruktion des Gleichungssystems
2. Lösen des Gleichungssystems
3. Auswertung der Interpolationsfunktion

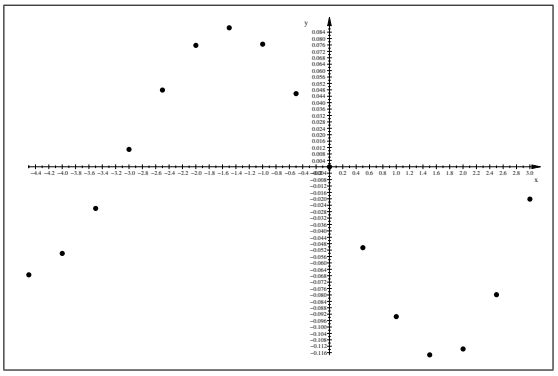
Der Aufwand für das Erstellen und Speichern des Gleichungssystems ist $O(n^2)$, da die Lösungsmatrix immer quadratisch ist.
Der Aufwand für das Lösen des Gleichungssystems hängt von der gewählten Methode ab. Wie man später sehen wird, erfolgt die Lösung des Gleichungssystems in dieser Arbeit mittels der QR-Zerlegung, welche laut Wikipedia [26] den doppelten Aufwand des Gaußverfahrens hat, also ca. $O(\frac{4}{3}n^3)$, dafür aber numerisch stabiler ist.
Zum Schluss muss die Approximationsfunktion d ausgewertet werden. Mit ihrer Hilfe wird ein Distanzbild erstellt, wie es im Abschnitt *Die Distanzfunktion* beschrieben ist. Es muss also für alle relevanten Pixel der Distanzwert berechnet werden. Der Aufwand dafür liegt bei $O(n)$, da die interpolierte Funktion aus n Linearkombinationen der RBF besteht.
Man beachte, dass für N gegebene Konturrandpunkte aufgrund der zusätzlichen Off-surface Points mit $3N$ Punkten gerechnet werden muss. Im folgenden Abschnitt kann gelesen werden, wie sich der Berechnungs- bzw. Speicheraufwand etwas reduzieren lässt.

Reduzierung des Berechnungsaufwands durch geschickte Wahl der Zentren

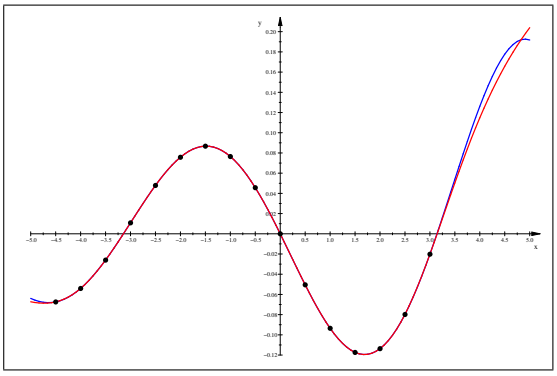
Durch eine geschickte Wahl der Zentren kann die Zahl der Punkte und damit der Berechnungsaufwand reduziert werden, ohne dass die Ergebnisqualität merklich beeinträchtigt wird [5]. Es soll nun das Beispiel in Abbildung 4.13 betrachtet werden.
Wie man erkennen kann, wurde die Funktion anfangs in 0,5er Schritten an 16 Stellen abgetastet und anschließend anhand dieser Zentren interpoliert (a - c). Bild (d) zeigt das Ergebnis, wenn man die Funktion lediglich an neun Stellen abtastet und dabei darauf achtet, dass der Kurvenverlauf ausreichend charakterisiert ist. Der Berechnungsaufwand ist so deutlich geringer, ohne dass merkliche Einbußen bei der Ergebnisqualität hingenommen werden müssen.



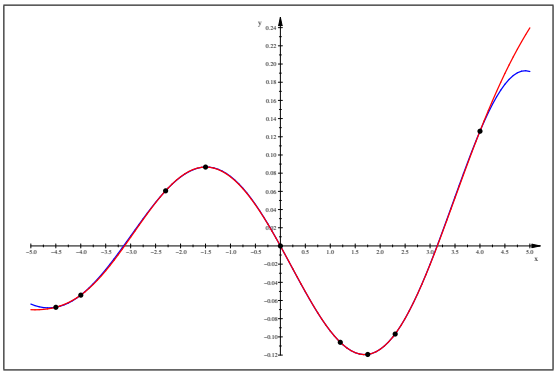
(a) Originalfunktion



(b) Abtastpunkte ohne Reduzierung



(c) Interpolierte Funktion (rot)



(d) Interpolierte Funktion (rot) nach Reduktion der Zentren

Abbildung 4.13: Reduktion der Zentren für die Interpolation

4.2.2 Implementierung

Für die Implementierung wurde die Interpolation in die folgenden Schritte aufgeteilt:

1. Extraktion der Konturrandpunkte
2. Reduzierung der Konturrandpunkte
3. Berechnung der Oberflächennormalen an den Konturrandpunkten
4. Interpolation der Distanzfunktion
5. Erzeugung des Distanzbildes
6. Erzeugung der Oberfläche
7. Erzeugung der 3D Segmentierung

In den nächsten Abschnitten wird nun die konkrete Implementierung der einzelnen Schritte beschrieben.

1. Extraktion der Konturrandpunkte

Als erster Schritt müssen die Randpunkte der eingezeichneten Konturen extrahiert werden. An diese wird später die Distanzfunktion approximiert. Diese Aufgabe wurde im `mitkImageToContourFilter` gelöst. Dieser benutzt den `itkContourExtractor2D-ImageFilter`, welcher auf Basis des *Marching Squares Algorithmus* [22, S.166-168] alle Konturen aus einem 2D Bild extrahiert.
Da die Konturen in beliebig orientierten Bildebenen eingezeichnet werden, kann es vorkommen, dass manche Schichtbilder Schnittkonturen aus anderen Schichten enthalten. Diese Schnittkonturen werden bei der Extraktion automatisch mit einbezogen. Innerhalb des betrachteten Volumens liegt dann eine schmale Kontur, die aufgrund eines Schnittes entstanden ist und zu einer falschen Interpolation führen würde. Deren Randpunkte würden, obwohl sie offensichtlich inmitten der Oberfläche liegen, die Distanz Null zugewiesen bekommen. Deshalb wird im nächsten Schritt nicht nur eine Reduktion der Anzahl der Konturrandpunkte durchgeführt, sondern es müssen zusätzlich diese Schnittkonturen eliminiert werden. Abbildung 4.14 veranschaulicht die Entstehung von Schnittkonturen.

2. Reduzierung der Konturrandpunkte

Wie aufgezeigt wurde, ist es nicht notwendig, jeden einzelnen Punkt mit in die Interpolation einzubeziehen. Um die Anzahl der Punkte zu reduzieren, wurde der `mitkReduce-ContourSetFilter` geschrieben. Dieser eliminiert gleichzeitig auch die vorhandenen Schnittkonturen, weshalb er alle existierenden Konturen als Input erhalten muss.
Bevor also eine Kontur reduziert wird, muss geprüft werden, ob es sich bei ihr um eine Schnittkontur handelt oder nicht. Diese Überprüfung erfolgt mittels einfacher Ebenenrechnung.

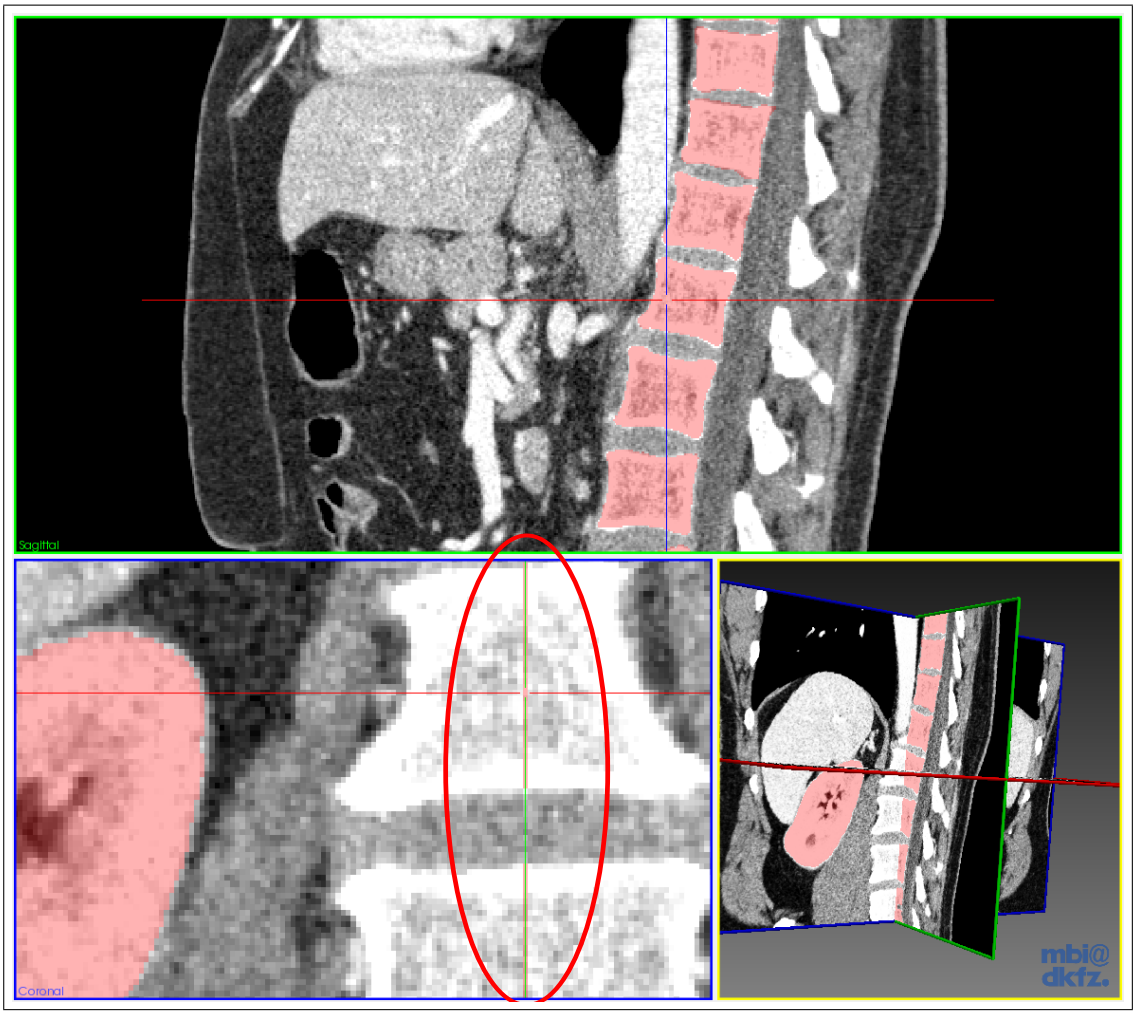


Abbildung 4.14: Die Entstehung von Schnittkonturen

Für jede Schicht, in der eine oder mehrere Konturen eingezeichnet wurden, wird auf die nachfolgende Weise eine Ebenengleichung in Normalenform aufgestellt.

Es werden drei Punkte $\vec{p_1}, \vec{p_2}$ und $\vec{p_3}$ einer Kontur der Schicht bestimmt. Mit diesen Punkten werden Vektoren erzeugt, die die Ebene aufspannen:

$$\vec{v_1} = \vec{p_2} - \vec{p_1}$$
$$\vec{v_2} = \vec{p_3} - \vec{p_1}$$

Aus den Vektoren wird über das Kreuzprodukt der Normalvektor der Ebene berechnet, der anschließend normiert wird:

$$\vec{n} = \vec{v_1} \times \vec{v_2}$$
$$\vec{n} = \frac{\vec{n}}{||\vec{n}||_2}$$

Daraus ergibt sich die Normalenform der Ebene:

$$\vec{n} \cdot \vec{x} = \lambda_E, \qquad \text{mit } \lambda_E = \vec{n} \cdot \vec{p_1}$$

Zu dieser Ebene wird nun der Abstand eines jeden Punktes der aktuell zu reduzierenden Kontur berechnet. Dafür wird die zuvor aufgespannte Ebene mit der Geraden g, die über den aktuellen Konturpunkt $\vec{p_k}$ und den Normalvektor der Ebene $\vec{n}$ definiert ist, geschnitten:

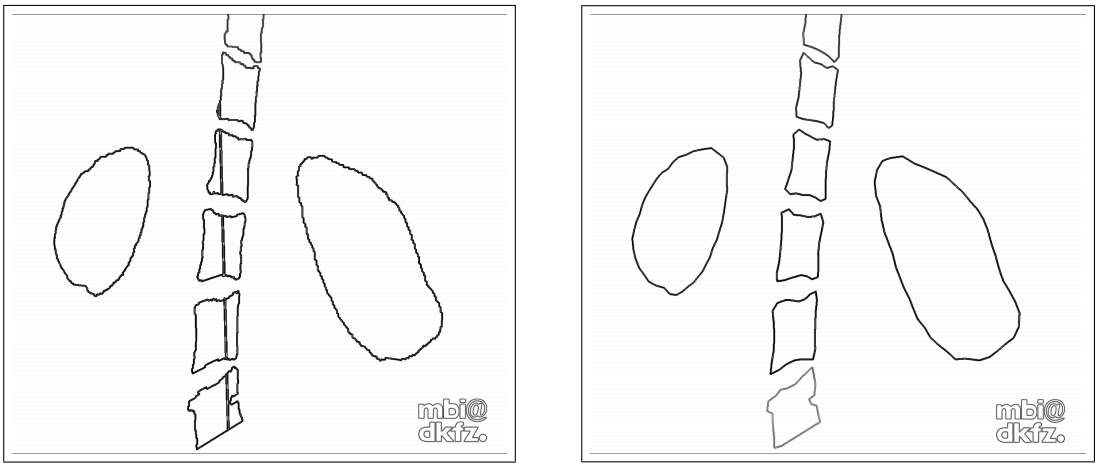
$$g = \vec{p_k} + \lambda_g \cdot \vec{n}$$
$$\lambda_E = \vec{n} \cdot (\vec{p_k} + \lambda_g \cdot \vec{n})$$

Der Abstand des Konturpunktes $\vec{p_k}$ ist damit:

$$\lambda_g = |\lambda_E - \vec{n} \cdot \vec{p_k}|$$

Jeweils der maximale und minimale Abstand werden zwischengespeichert. In Abhängigkeit des Pixel-Spacings werden für beide Abstandsmaße Schwellwerte festgelegt. Für den Minimalen liegt dieser bei dem 0,5-fachen des kleinsten, im Originalbild vorkommenden Spacings, und für den maximalen Abstand bei dem 1,5-fachen des größten Spacings des Bildes. Diese Grenzwerte wurden empirisch ermittelt. Liegen die jeweiligen Abstände unterhalb ihres Schwellwertes, handelt es sich um eine Schnittkontur, welche für die Reduktion vernachlässigt werden kann. Der soeben beschriebene Vorgang wird für jede Kontur durchgeführt. Abbildung 4.15 (a) zeigt die Schnittkonturen, die entstehen, wenn die in Abbildung 4.14 dargestellten Konturen extrahiert werden. In Abbildung 4.15 (b) kann das Ergebnis der Reduktion und Elimination gesehen werden.

Für die Reduktion wurden zwei alternative Algorithmen implementiert. Einer ist der *Nth-Point Algorithmus*. Bei diesem wird lediglich jeder n-te Punkt der Kontur zur Reduzierten hinzugenommen (Abbildung 4.16). Der große Nachteil dieser Methode ist, dass keinerlei Fehlermaß berücksichtigt wird und es deshalb vorkommen kann, dass markante Bereiche der Originalkontur ausgespart werden, wie das Beispiel in Abbildung 4.18 (b)



(a) (b)

Abbildung 4.15: Die Auswirkung von Schnittkonturen auf die Extraktion (a) und das Ergebnis derer Elimination (b)

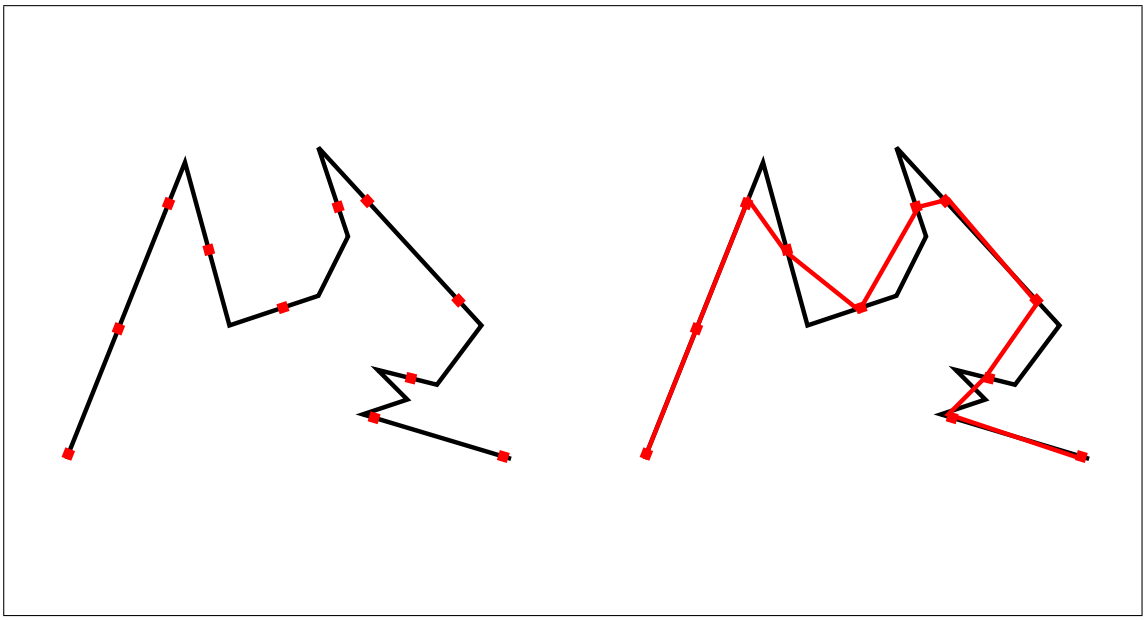


Abbildung 4.16: Die Vorgehensweise des Nth-Point Algorithmus

zeigt.

Als Zweites wurde der *Douglas-Peucker Algorithmus (DP)* [10] implementiert, wie ihn Ebisch [11] weiterentwickelt hat. Im Gegensatz zum Nth-Point wird beim DP der Fehler bei der Reduktion berücksichtigt. Die Fehlertoleranzgrenze kann dabei vom Benutzer vorgegeben werden. Die Kontur wird anfangs halbiert und es wird ein Vektor vom Startpunkt der Kontur zum Mittelpunkt erzeugt. Anschließend wird für jeden Punkt der Abstand zum Startvektor bestimmt. Der Punkt P_{max} mit dem größten Abstand wird zur Erzeugung zweier neuer Liniensegmente bzw. Vektoren benutzt (Start - P_{max} und P_{max} - Mitte).
Für jedes Segment wird der Punkt mit dem größten Abstand bestimmt. Mit diesem werden wiederum zwei neue Segmente erzeugt. Dies geschieht solange, bis der größte Abstand kleiner oder gleich der Toleranzschwelle ist und jeweils Start- und Endpunkt des aktuellen Segments in die Liste der reduzierten Punkte gespeichert werden. Abbildung 4.17 zeigt schematisch die Funktionsweise des DP Algorithmus. Das Ergebnis ist eine gute Approximation an die tatsächliche Kontur (Abbildung 4.18 (b)). An den reduzierten Konturrandpunkten müssen nun als nächsten Schritt die Oberflächennormalen approximiert werden.

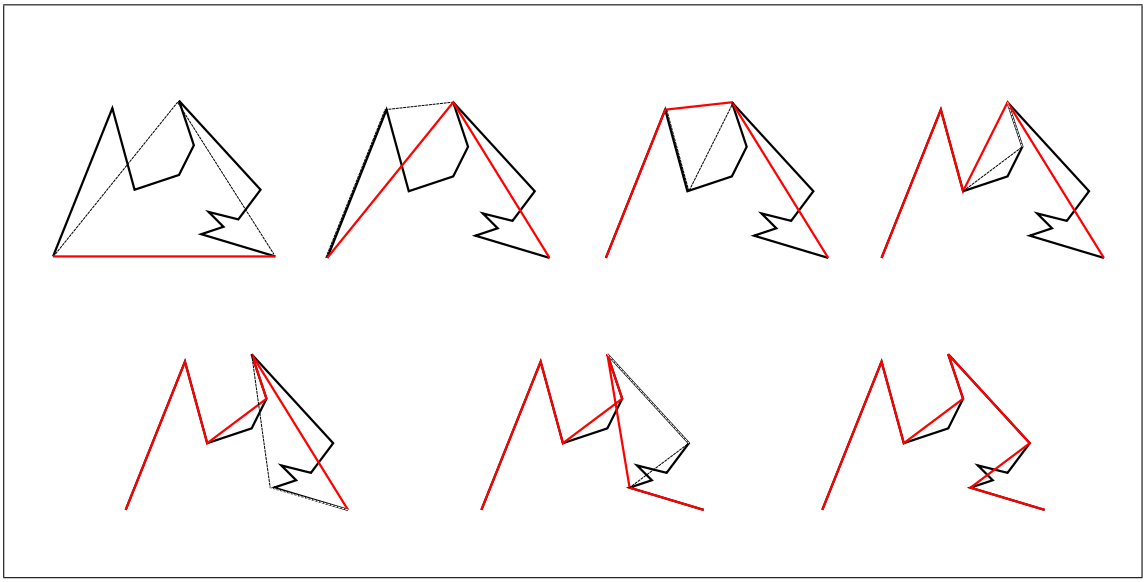
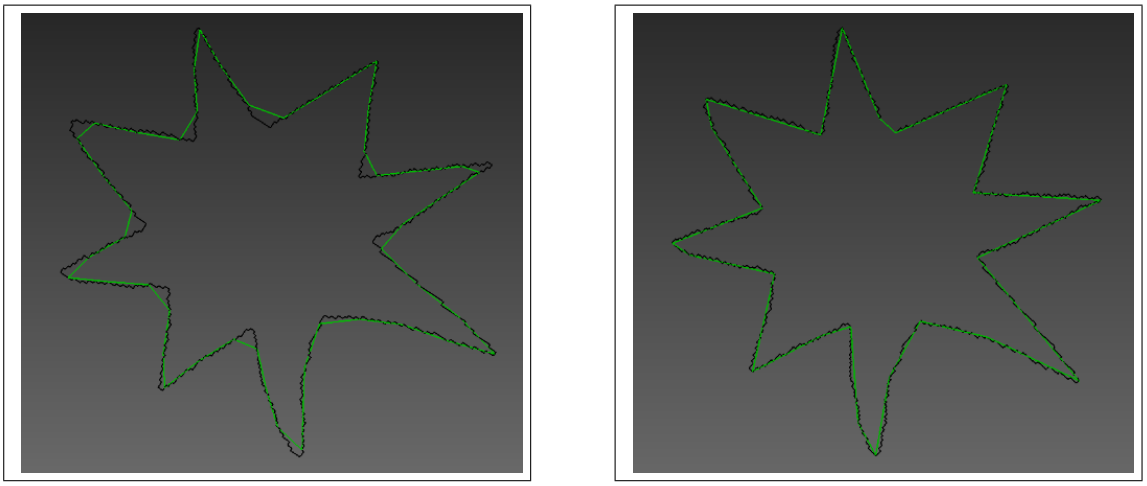


Abbildung 4.17: Die Vorgehensweise des Douglas-Peucker Algorithmus



(a) (b)

Abbildung 4.18: Beispiel einer Konturreduzierung mit (a) dem Nth-Point (Schrittweite = 40) und (b) dem Douglas-Peucker (Fehlertoleranz = 1,5)

3. Berechnung der Oberflächennormalen an den Konturrandpunkten

In Abschnitt *Bedeutung der Oberflächennormalen für die Berechnung* wurde erklärt, dass, um die Lösung Null zu vermeiden, für jeden Punkt zwei entsprechende Offsurface-Points erzeugt werden müssen. Das geschieht, indem die Oberflächennormale an dieser Stelle zu dem Punkt addiert bzw. subtrahiert wird. Da die Oberflächennormalen an den Stellen der Randpunkte nicht bekannt sind, müssen sie approximiert werden. Dazu wurde der `mitkComputeContourSetNormalsFilter` geschrieben.
Wichtig dabei ist, dass alle Normalen bzgl. der Konturen nach außen zeigen, um sicherzugehen, dass den jeweiligen Offsurface-Points später der korrekte Distanzwert zugeordnet wird. Die Berechnung der Normalen für jeden Punkt läuft folgendermaßen ab: Zuerst wird der Normalenvektor für die Ebene, in der die Kontur liegt, bestimmt. Dafür werden zwei Vektoren aus den Konturpunkten erstellt und für diese das Kreuzprodukt berechnet. Damit die Ebenennormale, und damit auch die Punktnormalen immer in dieselbe Richtung weisen, werden die zwei Konturvektoren nicht zufällig ausgewählt:
Liegt eine Kontur mit n Punkten vor, wird der erste Vektor zwischen den Punkten an der Stelle 0 und $\frac{n}{2}$ aufgespannt und der zweite Vektor zwischen den Punkten $\frac{n}{4}$ und $\frac{3}{4}n$. Wegen der rechten Handregel für das Kreuzprodukt zeigt die Ebenennormale dadurch bzgl. der Kontur immer in die gleiche Richtung.
Mit der Ebenennormalen wird anschließend an jedem Punkt die Normale der Oberfläche approximiert. Dafür werden drei aufeinanderfolgende Punkte benötigt (z.B. P_1,P_2,P_3). Wieder werden Vektoren erzeugt, und zwar zwischen P_1 und P_2 und zwischen P_2 und P_3. Für jeden Vektor wird wieder das Kreuzprodukt berechnet, dieses Mal aber mit der Ebenennormalen. Die beiden so erhaltenen Vektoren (in Abbildung 4.19 $\vec{a}$ und $\vec{b}$) werden addiert und gemittelt, um den approximierten Oberflächen-Normalenvektor ($\vec{n}$)

für den Punkt P_2 zu erhalten, der zum Schluss normiert wird. Die Berechnung wird in Abbildung 4.19 verdeutlicht.

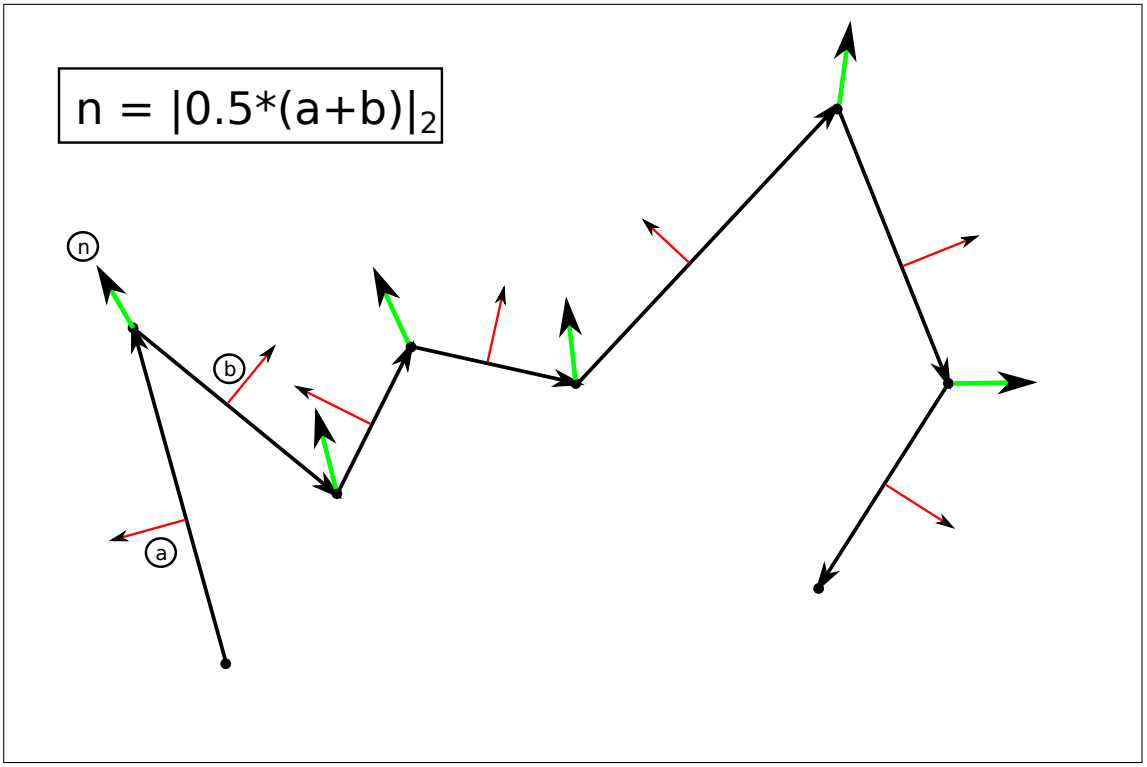


Abbildung 4.19: Vorgehen bei der Approximation der Oberflächennormalen: Die grünen Normalvektoren werden aus den roten gemittelt

Für die endgültige Bestimmung der Richtung der Normalen wird zwischen zwei Konturtypen unterschieden: *positive* und *negative* Konturen. Positiv sind Konturen, die von keiner anderen umgeben sind. Die berechneten Normalen zeigen bei ihnen vom Inneren der gesuchten Oberfläche weg. Wird aber aus einer eingezeichneten Kontur ein Loch herausgeschnitten, so werden dessen Ränder ebenfalls automatisch in Schritt 1 als Kontur extrahiert. In diesem Fall handelt es sich um eine negative Kontur, da sie innerhalb einer anderen liegt. Damit der berechnete Normalvektor, der wie es definiert wurde, in diesem Fall nicht in das von der Oberfläche umschlossene Volumen zeigt, muss dessen Richtung umgedreht werden. Zur Bestimmung der Richtung der Normalen für die aktuelle Kontur wird die Normale auf einen Punkt der Kontur addiert und im Segmentierungsbild der Pixelwert an dieser Stelle betrachtet. Ist an diesem eine Kontur eingezeichnet, dann ist die aktuelle Kontur negativ, andernfalls ist die Kontur positiv. Abbildung 4.20 veranschaulicht dieses Problem.

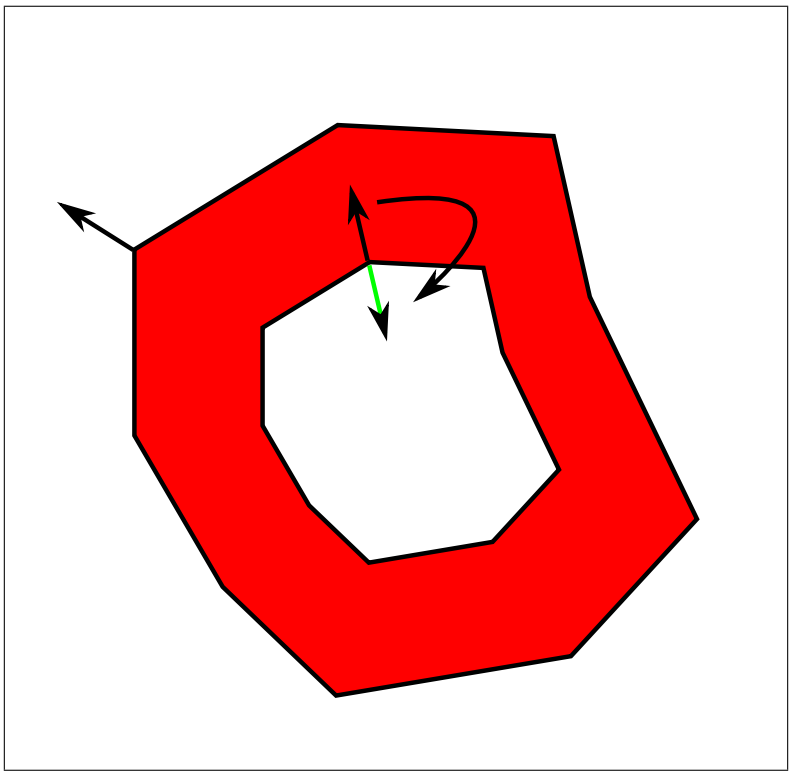


Abbildung 4.20: Eine Kontur, die von keiner anderen umgeben ist, ist eine positive Kontur und deren Normalen können wie berechnet übernommen werden. Eine negative Kontur liegt innerhalb einer anderen. Deren Normalen müssen umgedreht werden, damit sie nicht in das Innere der Oberfläche zeigen.

4. Interpolation der Distanzfunktion und Erzeugung des Distanzbildes

Mit den extrahierten Randpunkten und deren Normalvektoren hat man alle Informationen, um das Gleichungssystem für die Interpolation aufzustellen. Für die mathematische Repräsentation des Gleichungssystems und dessen Lösung wurde auf die VNL-Bibliothek[1] zurückgegriffen. Für diesen Arbeitsschritt wurde der `mitkCreateDistanceImageFromSurfaceFilter` erstellt. Dieser muss alle Konturen einschließlich deren Normalenvektoren erhalten. Daraus erzeugt er das Distanzbild, das letztlich den Verlauf der Oberfläche beschreibt.
Zuerst werden die Punkte aller Konturen in eine Liste gespeichert, `m_Centers` genannt. Anschließend werden mit Hilfe der Normalvektoren die Offsurface-Points erzeugt und ebenfalls der Punkteliste hinzugefügt. Analog dazu wird ein Vektor mit den entspre-

[1] http://vxl.sourceforge.net

chenden Funktionswerten gefüllt, also 0 für die Randpunkte und -1 bzw. 1 für die Offsurface-Points:

```
vnl_vector<double> m_FunctionValues;
```

Aus der Punkteliste wird die Lösungsmatrix erzeugt:

```
vnl_matrix<double> m_SolutionMatrix;
```

Mit `m_SolutionMatrix` und `m_FunctionsValues` ist das Gleichungssystem aus Gleichung 4.2 vollständig. Die Lösung erfolgt unter Zuhilfenahme des `vnl_qr<T>` Solvers:

```
vnl_qr < double > solver(m_SolutionMatrix);
vnl_vector < double > m_Weights = solver.solve(m_FunctionValues);
```

Dabei sind `m_Weights` die Interpolationsgewichte λ_i. Mit ihnen kann nun die Distanzfunktion, wie sie in Gleichung 4.1 definiert ist, aufgestellt werden.

5. Erzeugung des Distanzbildes

Bevor aber konkrete Distanzwerte berechnet werden können, muss erst ein leeres Distanzbild initialisiert werden. Wie das 2D Bild bei der Extraktion einer rotierten Ebene, wird auch das Distanzbild als ITK Bild erzeugt, allerdings in 3D. Das heißt, dass hier wieder dessen Größe und Pixel-Spacing angegeben werden müssen. Einfach gesagt, wird dann über dessen Pixel iteriert und für jeden einzelnen Pixel der entsprechende Distanzwert berechnet. Da für die Berechnung eines Wertes bei N vorhandenen Zentren der RBF jeweils N euklidische Distanzen, N Subtraktionen, N Additionen und N Multiplikationen durchgeführt werden müssen, wurden für die Initialisierung des Distanzbildes die folgenden Optimierungsmaßnahmen ergriffen:

Begrenzung des Distanzbildvolumens

Die Pixelanzahl des Bildes richtet sich einerseits nach dessen Größe, andererseits nach dessen Pixel-Spacing. Daher wurde die Größe des Distanzbildes an die Größe der Segmentierung im Originalbild angepasst. Anhand der eingezeichneten Konturen wird eine Bounding Box für die vorhandenen Konturen erstellt, welche gleichzeitig die Ausdehnung des Distanzbildes in Millimeter vorgibt. Seien

$$x_{min},\ y_{min},\ z_{min},\ x_{max},\ y_{max},\ z_{max}$$

die Grenzen der eingezeichneten Konturen,

so ergibt sich als Ausdehnung der gesuchten Oberfläche:

$$extent_x = x_{max} - x_{min}$$
$$extent_y = y_{max} - y_{min}$$
$$extent_z = z_{max} - z_{min}$$

Die Auflösung wurde dann unabhängig vom Originalbild festgesetzt. Dazu kann der Anwender zur Laufzeit das Volumen des Distanzbildes bestimmen (z.B. auf 500.000 Pixel). Um die vorgegebene Pixelanzahl zu gewährleisten, muss für die Ausdehnung und das Volumen ein entsprechendes Spacing berechnet werden (Das Spacing soll dabei in alle Richtungen gleich sein, also isotrop).

Die Formel für das Volumen lautet:

$$V = extent_x * spacing_x * extent_y * spacing_y * extent_z * spacing_z$$

Da das Spacing für das Distanzbild - wie bereits erwähnt - isotrop sein soll, ergibt sich als Formel für die Berechnung des Pixel-Spacings:

$$V = extent_x * extent_y * extent_z * spacing^3$$

$$spacing = \sqrt[3]{\frac{V}{extent_x * extent_y * extent_z}}$$

Auf diese Weise ist die Anzahl der Pixel, für die ein Distanzwert berechnet werden muss, unabhängig von der Größe und dem Pixel-Spacing des Originalbildes und unabhängig von der Größe des segmentierten Bereiches. Die Dauer hängt somit lediglich noch von der Anzahl der Punkte und damit von der Anzahl der Linearkombinationen der RBF ab, aus der die interpolierte Distanzfunktion besteht.

Narrow Band Auswertung der Distanzfunktion

Die Berechnung eines Distanzwertes läuft nun folgendermaßen ab: Jeder Pixel in Indexkoordinaten (`currentIndex`) des Distanzbildes wird in Weltkoordinaten transformiert. Da x_{min}, y_{min} und z_{min} den Ursprung des Distanzbildes und somit dessen Position im Weltkoordinatensystem angeben, kann die Transformation folgendermaßen berechnet werden:

```
Point3D currentPoint;
currentPoint[0] = currentIndex[0] * spacing + x_min;
currentPoint[1] = currentIndex[1] * spacing + y_min;
currentPoint[2] = currentIndex[2] * spacing + z_min;
```

Anschließend wird der `currentPoint` (in Weltkoordinaten) an die Distanzfunktion weitergereicht und so dessen Distanzwert berechnet:

```
double distanceValue(0);
Point3D p1;
Point3D p2;
double norm;
for (unsigned int i = 0; i < m_Centers.size(); i++)
{
    p1 = m_Centers.at(i);
    p2 = currentPoint - p1;
    norm = p2.two_norm();
    distanceValue = distanceValue + norm * m_Weights.get(i);
}
```

Der daraus resultierende Distanzwert wird an der entsprechenden Stelle im Distanzbild gesetzt. Auf diese Weise erhält man ein Bild, in dem alle Pixel, die außerhalb der gesuchten Oberfläche liegen, positive Werte und alle, die innerhalb der Oberfläche liegen, negative Werte annehmen. Je näher man der Oberfläche kommt, desto kleiner werden die Distanzwerte, bis sie an Stellen, an denen die Oberfläche verläuft, Null werden.
Für ein solches Bild wurde das Volumen zum Beispiel auf 500.000 Bildpunkte festgelegt. Das bedeutet aber immer noch, dass für 500.000 Pixel ein Distanzwert berechnet werden muss. Vor allem in Anbetracht der Tatsache, dass der Bereich, durch den die Oberfläche verläuft, aufgrund der Definition impliziter Funktionen sehr schmal ist, würden auf diese Weise viele unnötige Berechnungen durchgeführt werden. Es wäre deshalb ausreichend,

die Distanzfunktion nur innerhalb dieses schmalen Bandes auszuwerten. Man spricht in diesem Zusammenhang von einer *Narrow-Band-Methode*. Wu et al. [32] waren die ersten, die eine Auswertung der RBF entlang eines solchen Bandes durchführten. Sie bedienten sich dabei der *Octtree-Subdivision-Methode*. In dieser Arbeit wurde dagegen auf die Methode des Bereichswachstums (*Region Growing*) zurückgegriffen. Beginnend mit einem Pixel, durch den eine der Konturen verläuft, wird dieser in eine Liste gespeichert, um anschließend folgende Schritte durchzuführen:

1. Hole den nächsten Pixel aus der Liste (die anfangs nur den Startpunkt enthält).
2. Berechne für alle Pixel der 6er Nachbarschaft die Distanzwerte.
3. Jeder Pixel, dessen Distanzwert unterhalb eines Grenzwertes liegt, wird in die Liste gespeichert.
4. Beginne wieder mit Schritt 1, solange bis die Liste keine Punkte mehr enthält.

Empirisch hat sich als geeigneter Grenzwert das Doppelte des Spacings des Distanzbildes als sinnvoll erwiesen. Durch das *Region Growing* werden im Distanzbild nur die Pixelwerte berechnet, die sich in geringem Abstand zur Oberfläche befinden. In der Mitte dieses Bandes sollte die Distanz annähernd Null sein, also genau dort, wo die Oberfläche verläuft. Da die zuvor interpolierte Distanzfunktion die gesuchte Oberfläche implizit beschreibt, ist sichergestellt, dass alle relevanten Pixel mit einbezogen werden. Es muss allerdings garantiert werden, dass alle Pixelwerte außerhalb der Oberfläche positiv sind und alle innerhalb negativ. Dazu wird das Bild zuerst mit dem Pixelwert $+10$ für alle Pixel initialisiert. Anschließend erfolgt die Narrow-Band-Distanzberechnung. Als letzter Schritt wird allen Pixelwerten, die innerhalb des Bandes liegen, der Wert -10 zugewiesen. Abbildung 4.21 zeigt ein so erstelltes Distanzbild. Das rote Band veranschaulicht den Verlauf der Pixel, für die ein Distanzwert berechnet wurde.

6. Erzeugung der Oberfläche

Der Schritt vom Distanzbild zur Oberfläche ist einfach. Die VTK Bibliothek bietet dazu den *Marching Cubes Agorithmus* [22, S.168] an, ein Algorithmus speziell zu Erzeugung von Iso-Oberflächen aus 3D Bildern. Diesem wird das Distanzbild übergeben und als Schwellwert für die Pixel, durch die die Oberfläche verlaufen soll, wird entsprechend die Distanz 0 festgesetzt. Abbildung 4.22 zeigt das letztendliche Ergebnis in Form der interpolierten Oberfläche.

7. Erzeugung der 3D Segmentierung

Um schließlich von der Oberfläche wieder zu einer Segmentierung zu gelangen, bietet MITK eine entsprechende Funktionalität an. Mit dem Segmentation-Plugin lässt sich schnell zu einer gegebenen Oberfläche und einem gegebenen Bild eine Segmentierung erzeugen. Dabei werden alle Pixel markiert, die von der Oberfläche umschlossen werden.

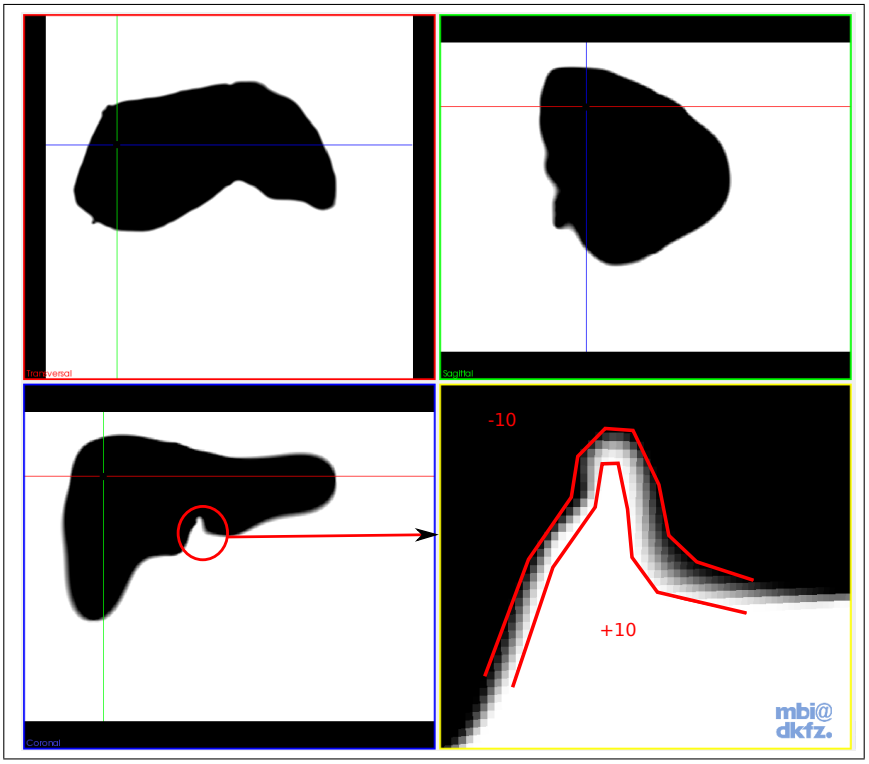
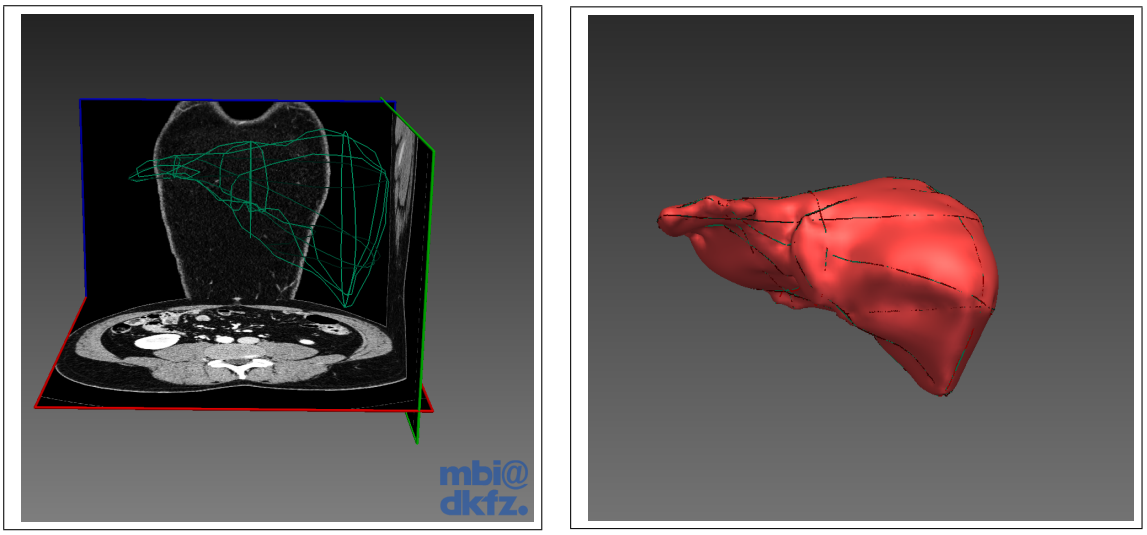


Abbildung 4.21: Das erzeugte Distanzbild. Das rote Band beschreibt den Verlauf der Narrow Band Distanzwertberechnung



(a) (b)

Abbildung 4.22: Die interpolierte Oberfläche (b), (a) zeigt die Position der eingezeichneten Konturen

5 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Arbeit vorgestellt. Das Kapitel ist dafür in folgende Bereiche untergliedert: Im ersten Teil wurden die für die Reduzierung der Konturen entwickelten Algorithmen evaluiert. Dabei wurde betrachtet, welche Auswirkungen die unterschiedlichen Methoden letztlich auf die interpolierte Oberfläche haben. Der Douglas-Peucker und der Nth-Point Algorithmus wurden jeweils für unterschiedliche Schrittweiten bzw. Fehlertoleranzwerte getestet.
Im zweiten Teil wurde untersucht, welchen Effekt die Benutzung von verschiedenen Volumina für das Distanzbild auf die Oberfläche hat.
Im dritten und letzten Teil wurde betrachtet, wie sich die Interpolation für unterschiedlichste Strukturen verhält, wenn die Anzahl der eingezeichneten Konturen variiert wird. Dazu wurde die 3D Segmentierung für die Leber, die Prostata und den Femur einer Ratte getestet. Außerdem wurde untersucht, wie sich der Einsatz von beliebig orientierten Bildebenen bei der Segmentierung auf die Anzahl der einzuzeichnenden Konturen auswirkt, um eine qualitativ gute Beschreibung der gesuchten Oberfläche zu erhalten.
Die Resultate der Segmentierungen unter Verwendung der verschiedenen Reduktionsmethoden und für variierende Anzahlen von Konturen, wurden anschließend mit den jeweiligen Experten-Segmentierungen verglichen. Für die Auswertung wurde das Programm, *EvaluateSegmentationResult* verwendet, welches für die von Heimann et al. [15] vorgestellte Studie entwickelt wurde. Alle Tests wurden auf einem 64 Bit Windows 7 Rechner mit einem Intel Core 2 Quad 2,4 GHz Prozessor und 4 GB Arbeitsspeicher durchgeführt.

Evaluationsmetriken für den Vergleich zweier Segmentierungen

Wie einleitend geschrieben, werden die Resultate der entwickelten interaktiven 3D Segmentierung mit entsprechenden Expertensegmentierungen verglichen. Dafür wird das Tool *EvaluateSegmentationResult* verwendet, das T. Heimann et al. für eine Vergleichsstudie existierender Segmentierungsalgorithmen entwickelt haben [15]. In ihrer Arbeit schreiben T. Heimann et al., dass die gebräuchlichsten Maße für den Vergleich zweier Segmentierungen solche sind, die auf volumetrischer Überlappung und Oberflächendistanzen basieren. In ihrer Studie kombinieren sie beide Ansätze, was den Vorteil mit sich bringt, dass so unterschiedliche Aspekte der Qualität einer Segmentierung verdeutlicht werden. Ein Teil der in der Studie verwendeten Maße soll auch in dieser Arbeit benutzt werden. Diese werden im Folgenden definiert.

Average Symmetric Surface Distance (Avg. Dist.):

Für zwei Segmentierungen A und B wird die Distanz derer Oberflächenpixel zueinander gemessen. Dazu wird die euklidische Distanz eines jeden Oberflächenpixels von A zum jeweils nahegelegensten aus B gemessen. Umgekehrt wird selbiges für B durchgeführt. Alle gemessenen Distanzen werden gespeichert und zum Schluss gemittelt.

Root Mean Square Symmetric Surface Distance (RMS Dist.):

Wie für die Avg. Dist., werden auch hier die euklidischen Distanzen berechnet. Allerdings werden diese quadriert, bevor sie gespeichert werden. Die RMS Dist. ergibt sich dann aus der Quadratwurzel des Mittelwertes aller berechneten (quadrierten) Distanzen. Der Vorteil gegenüber der Avg. Dist. ist, dass große Abweichungen von der Originaloberfläche härter bestraft werden. Auch die RMS Dist. wird in Millimeter gemessen.

Maximum Symmetric Surface Distance (Max. Dist.):

Sie wird wie die zwei vorhergegangenen Maße bestimmt. Allerdings wird hier nur die maximale gemessene Distanz in Millimetern zwischen zwei Segmentierungen A und B gespeichert.

Volumetric Overlap Error (Overlap Error):

Für zwei Segmentierungen A und B ist der Operlap Error folgendermaßen definiert:

$$100 \cdot \left(1 - \frac{|A \cap B|}{|A \cup B|}\right)$$

Er wird in Prozent angegeben.

Unabhängig von der vorgestellten Studie werden die folgenden eigenen Messungen durchgeführt:

Dauer: Die Zeit in Sekunden, die von der Interpolation der Distanzfunktion bis zur fertigen Oberfläche benötigt wird.

Anzahl Punkte: Die Anzahl der Konturrandpunkte, die in die Interpolation einfließen.

Maximale Segmentlänge (Max. SL): Bei der Benutzung des DP wird zusätzlich unter allen Konturen die größte vorkommende Segmentlänge in Anzahl Punkten gemessen.

Anzahl Konturen: Gibt an, wieviele Konturen für die Interpolation eingezeichnet wurden.

5.1 Die Algorithmen für die Konturreduktion

Es gibt mehrere Faktoren, welche die Qualität der interpolierten Oberfläche und damit der daraus resultierenden 3D Segmentierung beeinflussen. Einer davon ist der Algorithmus, mit dem die eingezeichneten Konturen reduziert werden. Die Randpunkte der Konturen sind gleichzeitig die Zentren der radialen Basisfunktionen, mit denen die Interpolation durchgeführt wird. Werden diese verändert, dann verändert sich auch das Resultat der Interpolation. Je nach gewählter Reduktionsmethode, spielen dabei auch deren Parameter eine wichtige Rolle. Im Falle des Nth-Point Algorithmus ist das die Schrittweite und beim Douglas-Peucker der Fehlertoleranzwert. Im Folgenden wurde der Einfluss der Reduktionsalorithmen auf das implementierte Verfahren unter Verwendung unterschiedlicher Parameter evaluiert.
Hierfür wurden 13 Konturen (Abbildung 5.1) aus einer bereits vorhandenen (Referenz-) Lebersegmentierung extrahiert. Die Anzahl der Randpunkte aller Konturen umfasste dabei insgesamt 23.769 Punkte. Das Volumen für das Distanzbild betrug in allen Fällen 1.000.000 Pixel

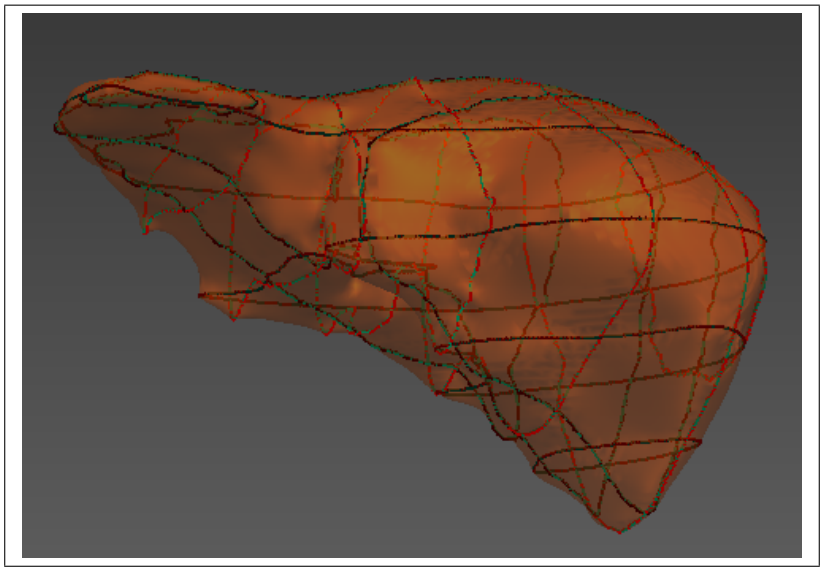
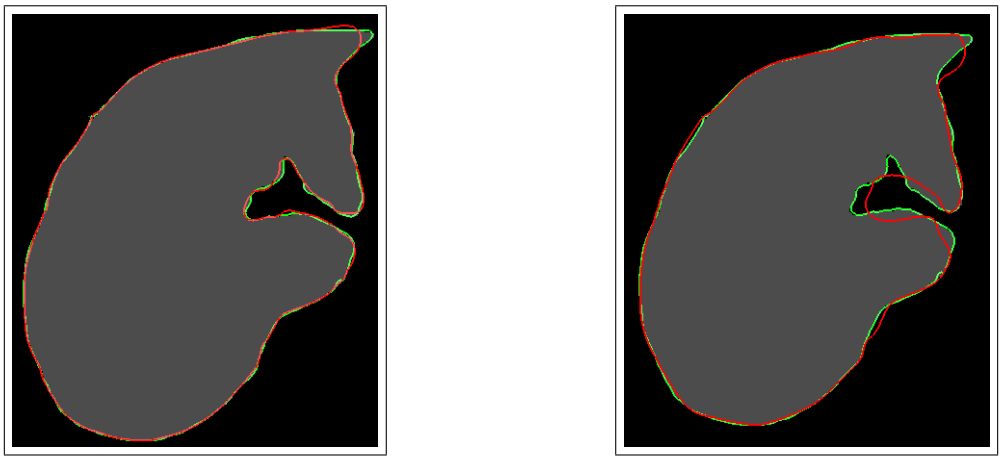


Abbildung 5.1: Position der Konturen, die aus der Referenzlebersegmentierung extrahiert wurden, um den Einfluss der unterschiedlichen Algorithmen für die Konturreduktion zu evaluieren. Diese Konturen wurden auch für die Auswertung unterschiedlicher Distanzbildvolumina verwendet

5.1.1 Der Nth-Point Algorithmus

Im ersten Schritt wurde der Nth-Point Algorithmus unter Benutzung unterschiedlicher Schrittweiten evaluiert. Die Ergebnisse werden in Tabelle 5.1 dargestellt. Wie sich eine Erhöhung der Schrittweite lokal bei komplexeren Konturen auswirkt, kann man in Abbildung 5.2 gut erkennen.



(a) $n = 20$ (grün) und $n = 50$ (rot) (b) $n = 20$ (grün) und $n = 120$ (rot)

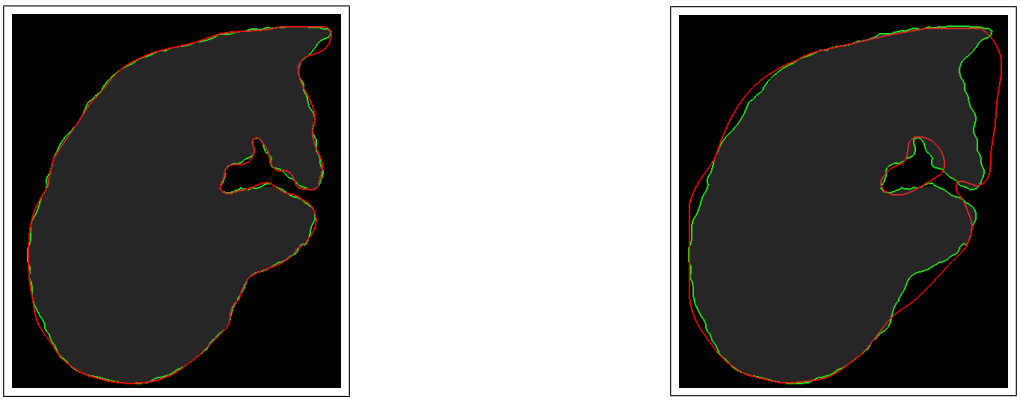
Abbildung 5.2: Vergleich der Auswirkung unterschiedlicher Schrittweiten n für den Nth-Point auf den Verlauf der interpolierten Oberfläche an Stellen, an denen Konturen (grau) eingezeichnet wurden

n	Anzahl Punkte	Dauer [s]	Avg. Dist. [mm]	RMS Dist. [mm]	Max. Dist. [mm]	Overlap Error [%]
20	1.210	110,8	0,45	0,73	5,86	4,55
50	494	21,0	0,53	0,90	11,41	5,23
80	316	13,2	0,65	1,05	11,12	6,18
120	218	10,1	0,82	1,29	11,81	7,63

Tabelle 5.1: Evaluierung des Einflusses unterschiedlicher Schrittweiten n für den Nth-Point Algorithmus auf die Oberflächeninterpolation. Gemessen wurden die Anzahl der Konturrandpunkte, die Dauer für die Erzeugung der Oberfläche und bzgl. der Referenzsegmentierung die mittlere Distanz, die Root Mean Square Distanz (RMS), die maximale Distanz und der volumetrische Überlappungsfehler.

5.1.2 Der Douglas-Peucker Algorithmus

Im Falle einer Reduzierung mit dem Douglas-Peucker Algorithmus muss ein Fehlertoleranzwert angegeben werden. Tabelle 5.2 zeigt die Ergebnisse für die Auswertung unterschiedlicher Toleranzwerte. Wie erwähnt wurden hier zusätzlich die jeweils größten Segmentlängen gemessen. Den Effekt höherer Fehlertoleranzwerte bei komplexeren Konturen zeigt Abbildung 5.3



(a) $tol = 0,5$ (grün) und $tol = 1,5$ (rot)

(b) $tol = 0,5$ (grün) und $tol = 10$ (rot)

Abbildung 5.3: Vergleich der Auswirkung unterschiedlicher Fehlertoleranzwerte *tol* für den Douglas-Peucker auf den Verlauf der interpolierten Oberfläche an Stellen, an denen Konturen (grau) eingezeichnet wurden.

tol. [mm]	Anzahl Punkte	Max. SL [Anzahl Punkte]	Dauer [s]	Avg. Dist. [mm]	RMS Dist. [mm]	Max. Dist. [mm]	Overlap Error [%]
0,5	1574	113	222,2	0,49	0,85	6,53	5,04
1,0	554	249	27,9	0,61	0,97	6,37	6,02
1,5	378	344	18,3	0,70	1,04	6,34	6,85
3,0	250	344	12,8	0,91	1,42	10,06	8,53
7,0	152	493	10,9	1,39	2,10	10,54	12,23
10,0	118	651	8,4	1,96	2,86	13,32	16,15

Tabelle 5.2: Evaluierung unterschiedlicher Fehlertoleranzwerte *tol* für den Douglas-Peucker Algorithmus. Gemessen wurden die Anzahl der Konturrandpunkte, die Dauer für die Erzeugung der Oberfläche und bzgl. der Referenzsegmentierung die mittlere Distanz, die Root Mean Square Distanz (RMS), die maximale Distanz und der volumetrische Überlappungsfehler. Außerdem wurde jeweils die größte vorkommende Segmentlänge (*Max. SL*) gemessen.

5.1.3 Vergleich des Douglas-Peucker mit dem Nth-Point Algorithmus

Abschließend wurde untersucht, wie sich der Douglas-Peucker gegenüber dem Nth-Point verhält. Visuell dargestellt wird die Auswirkung der unterschiedlichen Schrittweiten und Fehlertoleranzwerte für die beiden Reduzierungsalgorithmen in Abbildung 5.6. Dazu wurde für die erzeugten Oberflächen jeweils die Distanz bezüglich der Oberfläche der Referenzsegmentierung gemessen und visualisiert. Verwendet wurde dazu das Surface-Measurement Plugin von MITK. In der linken Spalte erfolgte die Reduzierung mit der Nth-Point Methode und in der rechten Spalte mit dem Douglas-Peucker Algorithmus.
Eine genauere Sicht liefert Abbildung 5.4, die den Douglas-Peucker (im Bild die rote Kontur) mit einer Fehlertoleranz von $0,5$ dem Nth-Point (im Bild grün) mit einer Schrittweite von 20 gegenüberstellt. Einen Vergleich des Douglas-Peucker mit $tol = 0,5$ und des Nth-Point mit $n = 50$ zeigt Abbildung 5.5.

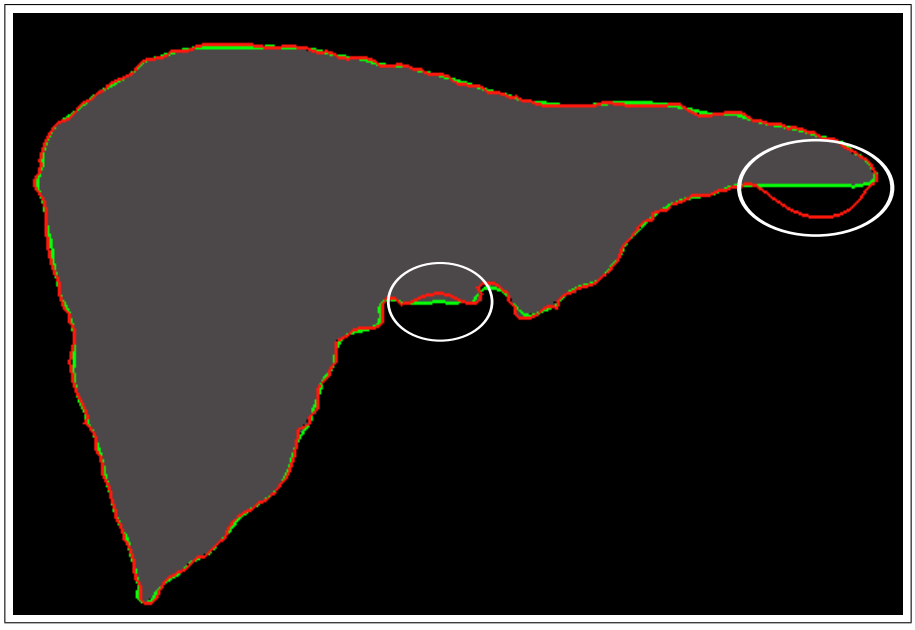


Abbildung 5.4: Vergleich der auf Basis des Douglas-Peucker (rot) mit $tol = 0,5$ und des Nth-Point (grün) mit $n = 20$ interpolierten Oberfläche anhand einer Leberkontur. Die weißen Kreise markieren größere Abweichungen.

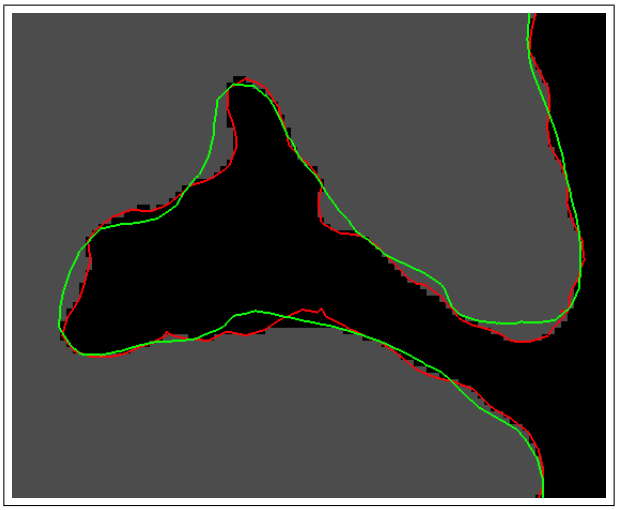


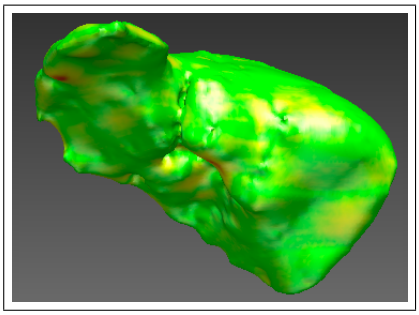
Abbildung 5.5: Vergleich der auf Basis des Douglas-Peucker (rot) mit $tol = 0,5$ und des Nth-Point (grün) mit $n = 50$ interpolierten Oberfläche anhand eines Ausschnittes einer Leberkontur

5.2 Unterschiedliche Distanzbildvolumina

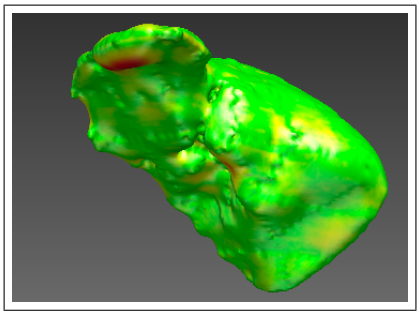
Die Qualität der interpolierten Oberfläche hängt auch von der Auflösung des Distanzbildes ab. Deshalb wurde im nächsten Schritt dessen Auflösung in Form des Volumens variiert. Für die Reduktion der Konturen wurde hier die Nth-Point Methode mit einer Schrittweite von 20 gewählt. Verglichen wurde wieder mit der Referenzsegmentierung. Die Ergebnisse finden sich in Tabelle 5.3

Volumen	Dauer [s]	Avg. Dist. [mm]	RMS Dist. [mm]	Max. Dist. [mm]	Overlap Error [%]
10.000	85,8	0,57	0,88	7,27	5,59
50.000	87,7	0,47	0,76	6,29	4,77
100.000	90,0	0,46	0,75	6,06	4,66
250.000	95,3	0,45	0,73	5,99	4,60
500.000	100,9	0,45	0,73	5,98	4,57
1.000.000	110,8	0,45	0,73	5,86	4,55

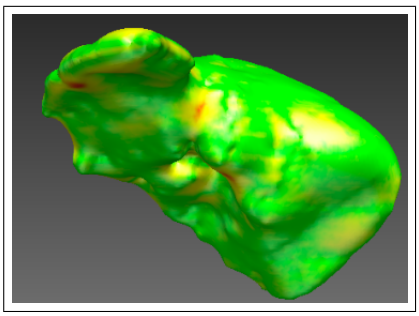
Tabelle 5.3: Evaluierung unterschiedlicher Volumina für das Distanzbild. Gemessen wurde die Dauer für die Erzeugung der Oberfläche und bzgl. der Referenzsegmentierung die mittlere Distanz, die Root Mean Square Distanz (RMS), die maximale Distanz und der volumetrische Überlappungsfehler



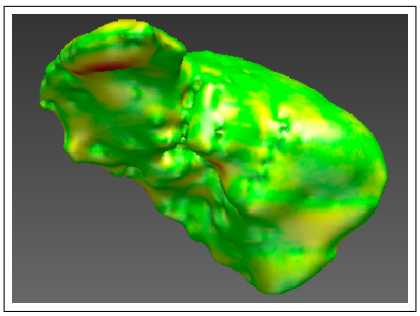
(a) $n = 20$



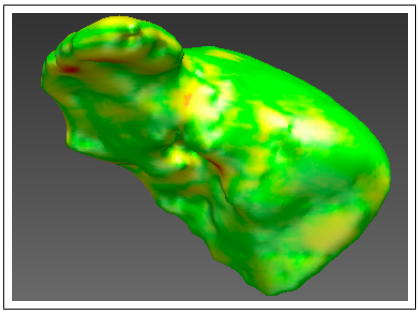
(b) $Tol = 0,5$



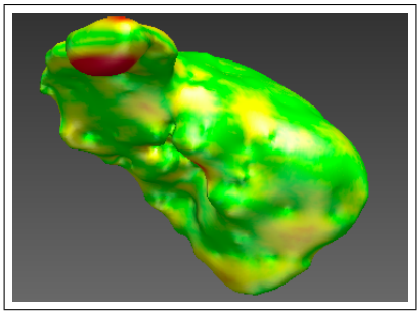
(c) $n = 50$



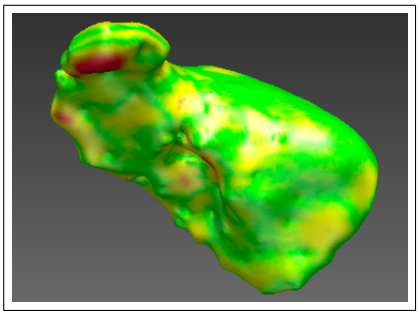
(d) $Tol = 1,5$



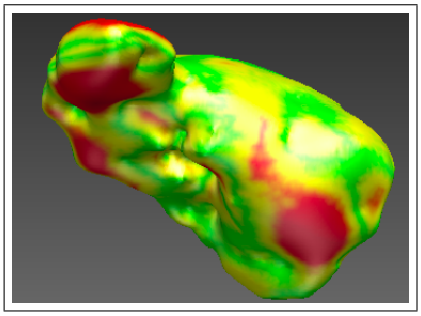
(e) $n = 80$



(f) $Tol = 3,0$



(g) $n = 120$

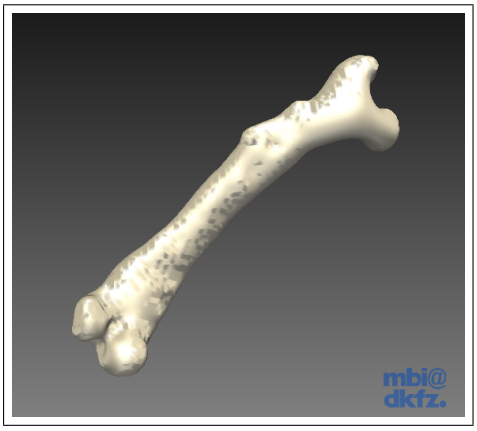


(h) $Tol = 10,0$

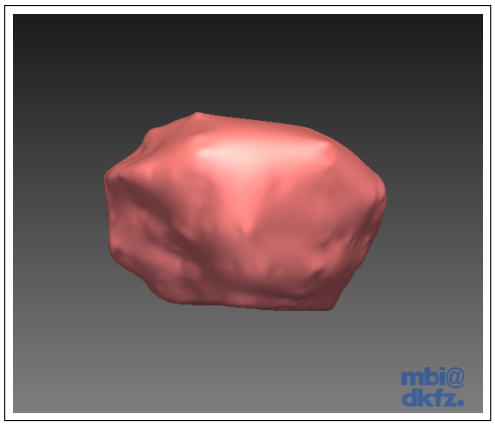
Abbildung 5.6: Distanzvisualisierung für unterschiedliche Parameterwerte. Links für den Nth-Point und rechts für den Douglas-Peucker Algorithmus. Grüne Bereiche haben dabei eine geringe Distanz zur Referenzoberfläche und gelbe bzw. rote Bereiche, eine größere.

5.3 Unterschiedliche Anzahl eingezeichneter Konturen

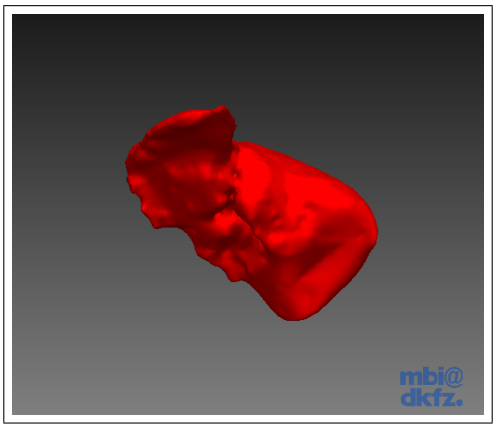
In diesem Abschnitt wird der Einfluss der Anzahl der eingezeichneten Konturen auf die Interpolation untersucht. Segmentiert wurden dafür die Leber, die Prostata und der Femur einer Ratte (siehe Abbilung 5.7). Für jede Struktur wurde die Anzahl der Konturen variiert. Auch hier wurden die erzielten Resultate wieder mit entsprechenden Expertensegmentierungen verglichen. Für die Auswertung wurde, wie in den vorigen Abschnitten, das *EvaluateSegmentationResult* Tool verwendet. Die Reduktion wurde mit dem Nth-Point Algorithmus mit $n = 20$ durchgeführt und das Volumen des Distanzbildes auf 500.000 gesetzt.



(a) Rattenfemur



(b) Prostata



(c) Leber

Abbildung 5.7: Strukturen für die Evaluation der 3D Segmentierung

Die Anzahl der Konturen zur Interpolation wurde sukzessive erhöht, wobei bestehende Konturen konstant verwendet wurden. In der nächsten Iteration wurde versucht, Bereiche zu finden, in denen die zuvor erzeugte Oberfläche stark von der Referenzsegmentierung abweicht. Dort wurden dann die nächsten Konturen eingezeichnet. Abbildung 5.8 zeigt einen solchen Bereich.

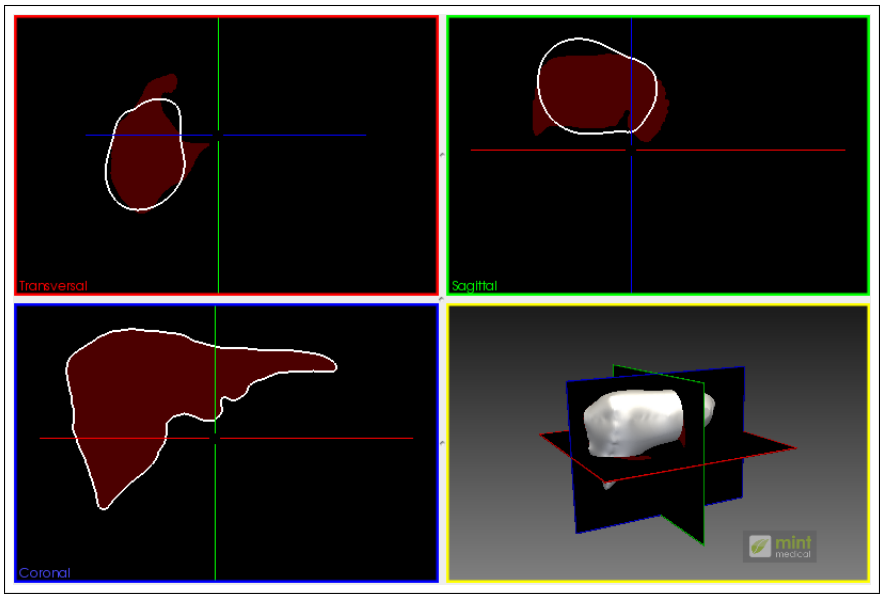
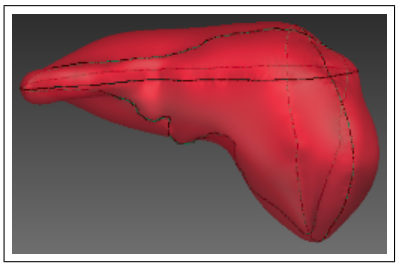
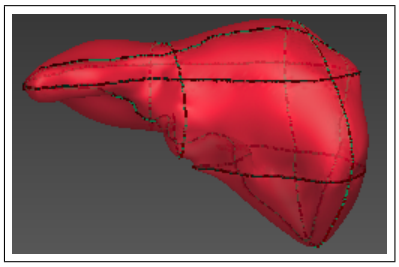


Abbildung 5.8: Setzen neuer Konturen in Bereiche, in denen die interpolierte Oberfläche noch sehr ungenau ist. Die roten Flächen stellen die Expertensegmentierung dar, die weiße Kontur gibt den Verlauf der interpolierten Oberfläche an.

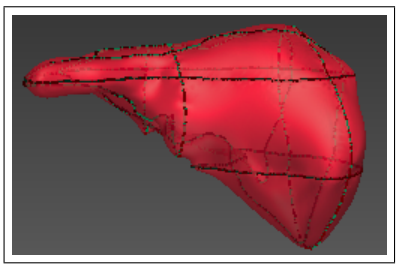
Eines der Ziele dieser Arbeit war, dass wenige Konturen für eine qualitativ gute Beschreibung einer Oberfläche ausreichend sein sollen. Dazu wurde unter anderem die Segmentierung in beliebig orientierten Bildebenen realisiert. Mit Ausnahme des Rattenfemurs wurden alle Strukturen anfangs in den Standardebenen segmentiert. Am Ende wurde versucht, mit der eben genannten Methode anhand möglichst weniger Konturen, wobei auch rotierte verwendet wurden, ein möglichst gutes Ergebnis zu erzielen. Aufgrund seiner Lage im Bild, siehe Abbildung 5.11, wurde der Rattenfemur nur zu beginn unter Verwendung der Standardebenen segmentiert. Im Anschluss daran wurden nur noch rotierte Schichten verwendet. An welchen Stellen die Konturen in den segmentierten Strukturen jeweils eingezeichnet wurden, ist in den Abbildungen 5.9 bis 5.12 zu sehen. Die Tabellen 5.4 - 5.6 enthalten die Ergebnisse.



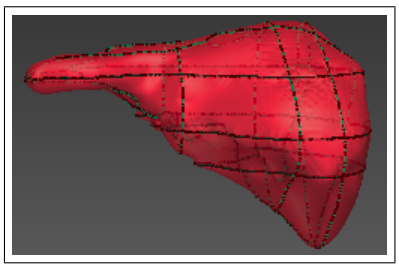
(a) 3 Konturen



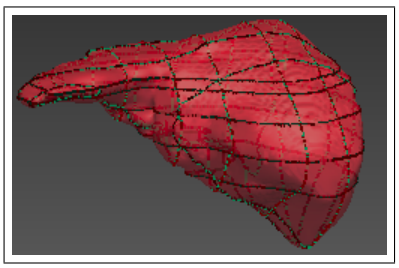
(b) 5 Konturen



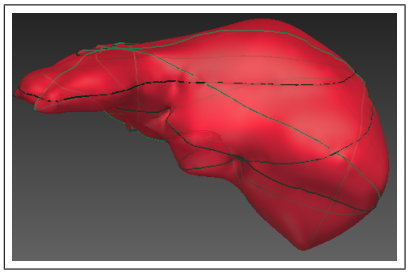
(c) 6 Konturen



(d) 8 Konturen



(e) 14 Konturen

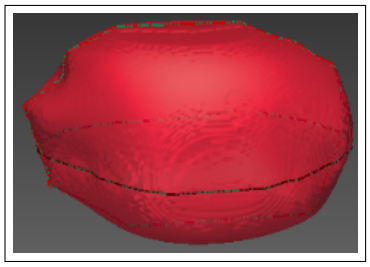


(f) 6 Konturen mit Rotation

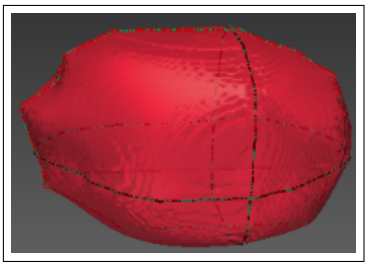
Abbildung 5.9: Lage der eingezeichneten Konturen für die Lebersegmentierung

Anzahl Konturen	Anzahl Punkte	Dauer [s]	Avg. Dist. [mm]	RMS Dist. [mm]	Max. Dist. [mm]	Overlap Error [%]
3	362	9,9	2,87	4,06	14,26	22,52
5	530	17,4	1,84	3,01	14,46	15,56
6	629	22,7	1,24	2,11	10,64	11,08
6 (r)	726	32,9	1,06	1,76	9,50	9,65
8	860	43,0	0,71	1,19	9,81	6,74
14	1377	136,0	0,38	0,65	6,17	4,01

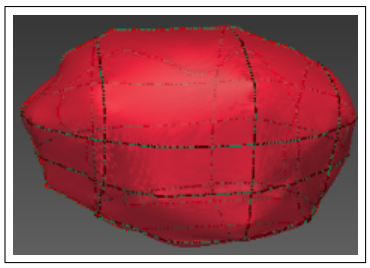
Tabelle 5.4: Evaluierung der 3D Segmentierung für die Leber bei unterschiedlicher Anzahl eingezeichneter Konturen. *(r)* gibt an, dass auch rotierte Konturen eingezeichnet wurden.



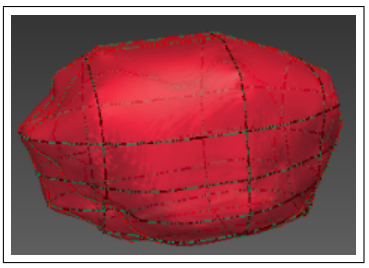
(a) 2 Konturen



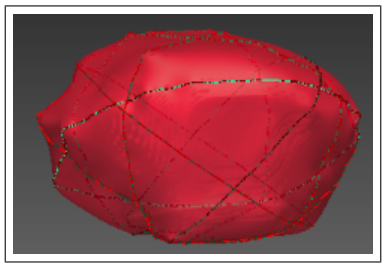
(b) 3 Konturen



(c) 8 Konturen



(d) 10 Konturen



(c) 6 Konturen mit Rotation

Abbildung 5.10: Lage der eingezeichneten Konturen für die Prostatasegmentierung

Anzahl Konturen	Anzahl Punkte	Dauer [s]	Avg. Dist. [mm]	RMS Dist. [mm]	Max. Dist. [mm]	Overlap Error [%]
2	231	4,9	0,97	1,44	5,31	15,40
3	324	7,5	0,54	0,84	3,59	9,21
6(r)	681	26,7	0,27	0,41	2,26	4,93
8	778	37,6	0,31	0,50	3,06	5,40
10	999	61,7	0,23	0,40	2,18	4,16

Tabelle 5.5: Evaluierung der 3D Segmentierung für die Prostata bei unterschiedlicher Anzahl eingezeichneter Konturen. *(r)* gibt an, dass auch rotierte Konturen eingezeichnet wurden.

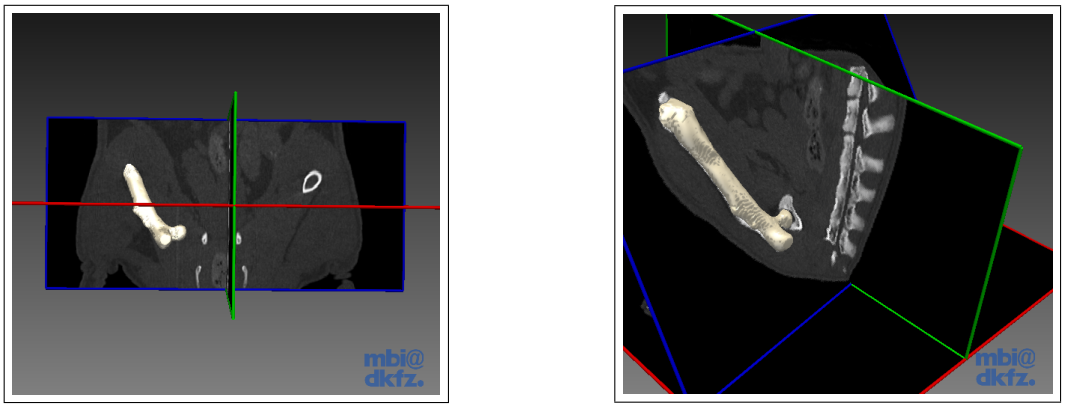
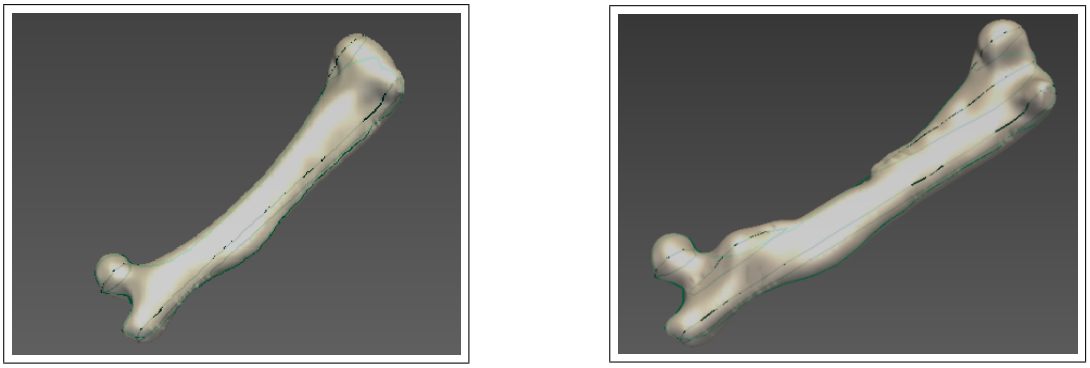
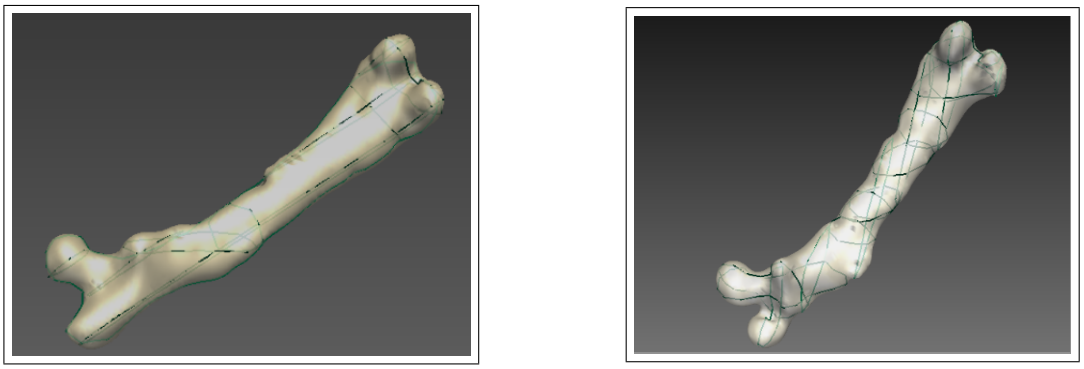


Abbildung 5.11: Lage des Rattenfemurs im Bild



(a) 3 Konturen mit Rotation

(b) 4 Konturen mit Rotation



(c) 8 Konturen mit Rotation

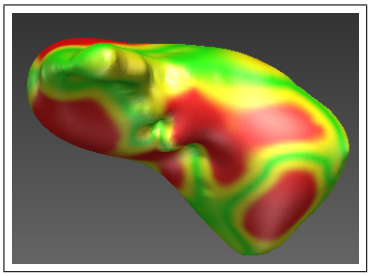
(b) 16 Konturen ohne Rotation

Abbildung 5.12: Lage der eingezeichneten Konturen für die Segmentierung des Rattenfemurs

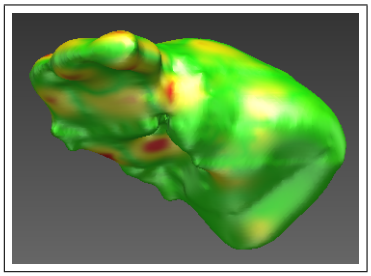
Anzahl Konturen	Anzahl Punkte	Dauer [s]	Avg. Dist. [mm]	RMS Dist. [mm]	Max. Dist. [mm]	Overlap Error [%]
3(r)	161	1,1	0,09	0,24	3,32	12,02
4(r)	206	1,4	0,08	0,17	1,12	11,20
8(r)	279	2,4	0,06	0,14	1,11	9,42
16	283	4,2	0,05	0,12	1,05	7,82

Tabelle 5.6: Evaluierung der 3D Segmentierung für den Femur einer Ratte bei unterschiedlicher Anzahl eingezeichneter Konturen. *(r)* gibt an, dass auch rotierte Konturen eingezeichnet wurden

In Abbildung 5.13 wird gezeigt, wie sehr sich die interpolierte Oberfläche ändert, wenn zusätzliche Konturen eingezeichnet werden. Untersucht wurde in diesem Experiment konkret die Veränderung der Oberfläche, wenn vor dem Hinzufügen weiterer Konturen entweder schon viele Konturen vorhanden waren oder nur einige wenige.



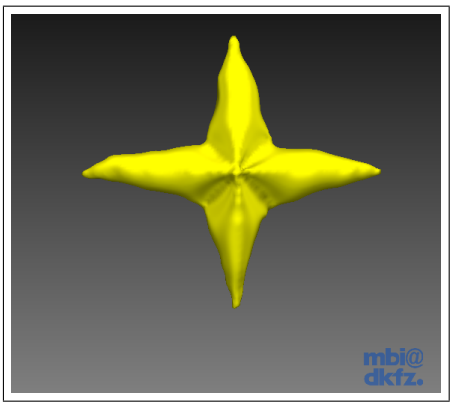
(a) Veränderung zwischen drei und acht Konturen für die Leber



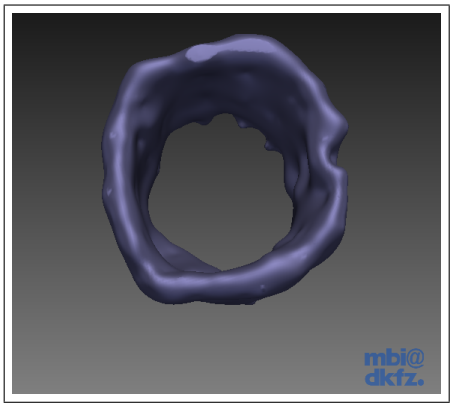
(b) Veränderung zwischen acht und vierzehn Konturen für die Leber

Abbildung 5.13: Auswirkung zusätzlicher Konturen auf die Leberoberfläche. Rote Bereiche markieren starke Änderungen.

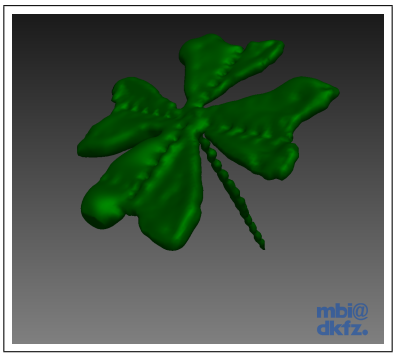
Um zu demonstrieren, dass mit dem entwickelten Verfahren jede beliebige Form interpoliert werden kann, wurde abschließend die Interpolation für ungewöhnliche Topologien getestet. Dazu wurden manuell die Konturen eines Kleeblattes, eines Sterns und einer Röhre in ein leeres Bild gezeichnet und anschließend eine Interpolation durchgeführt. Die Resultate zeigen die Tabelle 5.7 und Abbildung 5.14.



(a) Stern



(b) Röhre



(c) Kleeblatt

Abbildung 5.14: Verschiedene Formen zur Demonstration der Möglichkeiten des entwickelten Verfahrens

Form	Anzahl Konturen	Dauer [s]
Stern	5	5,9
Röhre	9	11,4
Kleeblatt	17	23,9

Tabelle 5.7: Evaluierung der Interpolation der Oberfläche für spezielle Formen

6 Diskussion

In Kapitel 5 wurde festgestellt, dass die Effizienz des entwickelten Verfahrens und damit die Qualität der interpolierten Oberfläche von mehreren Faktoren abhängt. Dazu wurden der Nth-Point Algorithmus mit unterschiedlichen Schrittweiten und der Douglas-Peucker mit verschiedenen Fehlertoleranzwerten evaluiert. Im Anschluss wurde die Auswirkung einer Veränderung des Distanzbildvolumens untersucht.
Abschließend wurde die 3D Segmentierung an der Leber, der Prostata und einem Rattenfemur getestet, wobei die Anzahl der eingezeichneten Konturen variiert wurde.
Im diesem Abschnitt werden nun die erhaltenen Ergebnisse diskutiert.

6.1 Evaluation der Algorithmen für die Konturreduktion

Die implementierten Methoden für die Reduktion der Konturrandpunkte und damit der Anzahl der Zentren, welche für die Interpolation verwendet werden, unterscheiden sich in ihrem Vorgehen deutlich. Während der Nth-Point einfach nur jeden n-ten Punkt in die reduzierten Konturen aufnimmt, wird beim DP ein Fehlermaß berücksichtigt. Es ist daher auf den ersten Blick verwunderlich, dass der Douglas-Peucker trotzdem schlechtere Ergebnisse liefert. Es werden nun die Ergebnisse der einzelnen Algorithmen genauer betrachtet.

6.1.1 Der Nth-Point Algorithmus

In Tabelle 5.1 wird ersichtlicht, dass der Nth-Point Algorithmus mit einer Schrittweite von 20 das genaueste Ergebnis liefert. Eine Vergrößerung der Schrittweite führt in erster Linie zu einem starken Anstieg der maximalen Distanz, da markante Bereiche stärker geglättet bzw. abgeschnitten werden. Dies lässt sich gut in Abbildung 5.2 erkennen. Während bei einer Schrittweite von $n = 20$ die Form der Kontur annähernd zu 100% erhalten bleibt, werden für $n = 50$ spitzere Ecken bereits merklich abgerundet. Noch deutlicher ist dies für $n = 120$ zu erkennen. Die Tabelle 5.1 zeigt, dass bei einem Sprung von $n = 20$ auf $n = 50$ die Qualität der interpolierten Oberfläche stark nachlässt. Ab $n = 50$ hat die Erhöhung der Schrittweiten einen wesentlich geringeren Effekt auf die resultierende Oberfläche. Das ist damit erklärbar, dass ab $n = 50$ die spitzeren Ecken der betrachteten Form bereits deutlich abgerundet sind (Abbildung 5.2 a). Eine weitere Erhöhung der Schrittweite wirkt sich dadurch weniger aus. Die Dauer der Interpolation dagegen verringert sich drastisch. Besonders beim Schritt von $n = 20$ auf 50 verkürzt sich die Dauer um fast 80%.
In Abbildung 5.6 sieht man anhand der roten Bereiche, dass für große Schrittweiten lokal

teilweise große Fehler auftreten. Möchte man also eine möglichst genaue 3D Segmentierung erhalten, empfehlen sich kleinere Schrittweiten. Stellt die interpolierte Oberfläche aber nur ein Zwischenergebnis dar, so reichen auch große Schrittweiten aus. Ein Beispiel dafür wäre die Arbeit von Wimmer et al. [27], in der eine grobe initiale Oberfläche berechnet wird, die dann im zweiten Schritt unter Einbindung von Bildinformationen verfeinert wird.
Es muss beachtet werden, dass die Anzahl der Punkte einer Kontur nicht nur von der Größe der Kontur selbst abhängt, sondern auch vom Pixel-Spacing des Bildes, aus dem die Konturen extrahiert werden. Segmentiert man also kleine Strukturen in Bildern mit einem großen Spacing, so empfiehlt es sich kleine Schrittweiten zu wählen. Umgekehrt ist auch eine größere Schrittweite ausreichend, wenn große Strukturen in Bildern mit kleinem Spacing segmentiert werden.

6.1.2 Der Douglas-Peucker Algorithmus

Ein Blick auf Tabelle 5.2 lässt erkennen, dass der DP sogar bei der sehr niedrigen Fehlertoleranzschwelle von 0,5 mm sichtbar schlechtere Resultate liefert, als der Nth-Point mit $n = 20$, obwohl beim DP mehr Punkte in die Interpolation eingeflossen sind. Um dieses Verhalten zu erklären, wurde zusätzlich für jeden Toleranzwert die größte vorkommende Segmentlänge in Anzahl von Punkten gemessen. Für eine Schrittlänge von n ist diese beim Nth-Point ebenfalls gleich n. Beim DP fällt sofort auf, dass schon bei einer Toleranzgrenze von 0,5 die größte Segmentlänge bereits 113 beträgt. Für $tol = 10$ beträgt diese sogar 651. Auf solch großen Bereichen bricht die Distanzfunktion zwischen zwei Stützstellen aus und verläuft dadurch nicht mehr entlang der Segmentierung, was dann zu größeren Fehlern bezüglich der Referenzdaten führt. Abbildung 5.4 veranschaulicht dieses Problem. Die grüne Kontur stellt den Verlauf der Oberfläche bei Verwendung der Nth-Point Methode mit $n = 20$ dar. Die rote Kontur beschreibt den Verlauf für den Douglas-Peucker mit $tol = 0,5$. In den eingekreisten Bereichen sieht man, dass auf dem geraden Stück die Kontur des DP ausbricht, da nicht genügend Punkte für die Interpolation vorhanden sind. Auch der starke Anstieg der gemessenen Distanzen bei einer Erhöhung des Toleranzwertes lässt sich mit der analog wachsenden Segmentlänge erklären. In Abbildung 5.6 sieht man, dass die resultierende Oberfläche ab einer Toleranz von drei, an vielen Stellen größere Abweichungen aufweist. Dies trifft auch auf glatte Bereiche zu, in denen aufgrund des hohen Toleranzwertes lange Segmente auftreten.

Trotz allem werden insbesondere bei Verwendung des DP mit $tol = 0,5$, wie in Abbildung 5.5 sehr gut gezeigt wird, feine Strukturen exakter beschrieben, welche dagegen bei der Nth-Point (im Bild mit $n = 50$) Methode geglättet werden.

Zusammenfassend kann gesagt werden, dass der Nth-Point global gesehen bessere Resultate erzielt, da bei diesem die Zentren für die RBF-Interpolation gleichmäßig verteilt sind. Der Douglas-Peucker dagegen ist darauf ausgelegt, die Punkte von Kurven und Polygonen so zu verringern, dass diese unter Berücksichtigung des Fehlers immer noch

möglichst genau dargestellt werden. Eine Gerade mit 1.000 Punkten würde beim DP auf exakt zwei Punkte reduziert werden, während der Nth-Point bei einer Schrittweite von 20, 50 Punkte liefern würde. Dafür beschreibt der Douglas-Peucker spitze Ecken und scharfe Kurven deutlich besser als der Nth-Point. Es wäre also ein guter Ansatz, beide Methoden zu kombinieren, indem darauf geachtet wird, dass die Lücken beim DP nicht zu groß werden.

6.2 Evaluation für unterschiedliche Distanzbildvolumina

Ein weiterer betrachteter Parameter war die Auflösung des Distanzbildes. Diese wird, wie bekannt ist, über das Volumen in Anzahl Pixeln angegeben. Die Tabelle 5.3 zeigt, dass große Volumina und damit hohe Auflösungen nur einen geringen Effekt auf die Ergebnisqualität haben und deshalb unnötig Rechenzeit kosten. Begründet werden kann das durch den Marching Cubes Algorithmus, mit dem die Oberfläche letztlich aus dem Distanzbild erzeugt wird. Dieser arbeitet im Subpixelbereich, weshalb eine niedrigere Auflösung eine nur sehr geringfügige Auswirkung hat. Die besten Ergebnisse konnten mit einem Volumen, das zwischen 50.000 und 500.000 Pixeln liegt, erzielt werden. Dies brachte in der Evaluation eine Zeitersparnis zwischen 10 und 23 Sekunden gegenüber dem maximalen getesteten Volumen von 1.000.000 ein. Auch hier gilt: Je größer der segmentierte Bereich in Millimeter ist, desto größer sollte auch das Volumen gewählt werden.
Verglichen mit einer Reduzierung des Volumens erzielt eine Reduzierung der Konturrandpunkte einen sehr viel größeren Effekt. Dadurch werden die Zentren für die Interpolation und damit die Anzahl der Unbekannten des zu lösenden Gleichungssystems verringert. Die Lösung des Gleichungssystems ist also deutlich aufwendiger als die Berechnung der Distanzwerte für viele Pixel.

6.3 Evaluation der 3D Segmentierung mit variierender Konturenanzahl

Im nächsten Schritt wurde die 3D Segmentierung für unterschiedliche Strukturen evaluiert, wobei jeweils die Anzahl der eingezeichneten Konturen variiert wurde. Es zeigt sich, dass bei komplexeren Formen, wie zum Beispiel der Leber, wesentlich mehr Konturen nötig sind, um gleichwertige Resultate zu erzielen. In Tabelle 5.4 sieht man, dass für die Leber 14 Konturen nötig sind, um einen Überlappungsfehler von ca. 4% zu erhalten, wohingegen für die Prostata bereits 10 Konturen dafür ausreichend sind (siehe Tabelle 5.5). Anhand der Prostata ist ersichtlich, dass für einfache Formen schon mit wenigen Konturen ein gutes Ergebnis erzielt werden kann. Drei Konturen der Prostata reichen aus, um einen Überlappungsfehler von gerade einmal 9,2% zu erreichen. Bei der Leber dagegen beträgt der Überlappungsfehler für drei Konturen 22,5%.
Des Weiteren wird ersichtlich, dass sich zusätzliche geschickt platzierte Konturen deutlich auf die Ergebnisqualität auswirken. Beispielsweise reduziert sich der Überlappungsfehler

für die Leber von 22,5% auf 15,5%, wenn statt nur drei Konturen fünf eingezeichnet werden. Andererseits wirkt sich das Platzieren zusätzlicher Konturen deutlich geringer aus, je mehr bereits eingezeichnet sind. Verringert sich der Überlappungsfehler beim Sprung von drei auf fünf Konturen noch stark, so hat der Sprung von acht auf vierzehn Konturen einen wesentlich kleineren Effekt auf den Überlappungsfehler. Abbildung 5.13 zeigt, wie stark sich die Leberoberfläche bei einem Sprung von drei auf acht Konturen (a) ändert und wie wenig sie sich beim Sprung von acht auf vierzehn Konturen (b) ändert.

Die Segmentierung in beliebig orientierten Bildebenen

Anhand der Segmentierung des Rattenfemurs wird ersichtlich, dass das Einzeichnen von Konturen in rotierten Bildebenen nötig ist, um auch Objekte, die schräg in einem Bildvolumen liegen, mit wenigen Konturen beschreiben zu können. Abbildung 5.12 zeigt, dass man wesentlich mehr Konturen in den Standardebenen einzeichnen muss, um eine gleichwertige Beschreibung der gesuchten Struktur zu erreichen. Aus Tabelle 5.6 lässt sich ablesen, dass schon drei Konturen ausreichen, um eine ziemlich genau 3D Segmentierung des Femurs zur erstellen. Dagegen müssen ganze 16 Konturen erstellt werden, um den Femur vollständig zu beschreiben. In rotierten Schichten ist in etwa die Hälfte ausreichend, um ein gleichwertiges Ergebnis zu erzielen.
Auch die Tabellen 5.4 und 5.5 zeigen am Beispiel der Leber und der Prostata, dass rotierte Ebenen mehr Freiheit in der Beschreibung der zu interpolierenden Oberfläche bieten. Vergleicht man die Ergebnisse mit den Tabellen 5.4 und 5.5, erkennt man, dass mit wenigen rotierten Konturen Resultate erreicht werden, für die in den Standardebenen noch zusätzliche Konturen eingezeichnet werden müssten. Der Grund dafür ist, dass markante Bereiche der gesuchten Oberfläche auch schräg im Bild liegen können und deshalb mit einer oder wenigen rotierten Konturen schnell beschrieben werden können.

Die Flexibilität des Verfahrens

Abschließend wurde versucht, in einem leeren Bild Konturen zu zeichnen, die eher ungewöhnliche Formen beschreiben. Damit sollte gezeigt werden, dass das Verfahren mit beliebigen Objekten schnell und stabil läuft und in solchen Fällen gute Oberflächen erstellt. Anhand eines Sterns, einer Röhre und eines Kleeblattes konnte das erfolgreich demonstriert werden. Besonders fällt hier auf, wie schnell selbst die komplexe Form des Kleeblattes interpoliert wird.

7 Schlussfolgerung und Ausblick

Die Aufgabenstellung war es, einen Algorithmus für eine schnelle und interaktive Erstellung einer 3D Segmentierung für das Framework MITK zu entwickeln. Um eine möglichst große Freiheit in der Beschreibung der gesuchten Oberfläche zu haben, sollten die vorhandenen manuellen 2D Segmentierungswerkzeuge so erweitert werden, dass sie auch in rotierten Bildebenen einsetzbar sind. Die Erstellung der 3D Segmentierung sollte über die Interpolation der Oberfläche erfolgen. Diese Ziele wurden erreicht.

Mit dem entwickelten Verfahren wird ein stabiler Algorithmus für die interaktive 3D Segmentierung medizinischer Strukturen bereitgestellt. Es können damit beliebige Formen beschrieben und interpoliert werden.

Zusätzlich ist die Interpolation dank der implementierten Optimierungen äußerst schnell. Mit einer geschickten Wahl des Distanzbildvolumens und einer guten Reduzierung der Konturen kann viel Zeit gespart werden, ohne größere Einbußen hinsichtlich der Qualität der Oberfläche hinnehmen zu müssen.

Durch die Verwendung der manuellen 2D Segmentierungswerkzeuge in MITK ist dem Benutzer ein hohes Maß an Interaktion gestattet. Relevante und markante Bereiche der untersuchten Struktur können dank beliebig orientierter Bildebenen schnell und mit wenigen Konturen beschrieben werden. Die Qualität der resultierenden Oberfläche hängt dabei sehr von der Anzahl der eingezeichneten Konturen ab.

Es existiert allerdings noch Raum für Weiterentwicklungen. Wie in Kapitel 6 beschrieben, würde sich eine Kombination der Algorithmen für die Reduzierung der Konturrandpunkte anbieten. Der Douglas-Peucker würde garantieren, dass alle markanten Bereiche einer Kontur beschrieben werden. Durch die Nth-Point Methode wären zudem die Zentren für die Interpolation gleichmäßiger verteilt, was, wie sich herausgestellt hat, einen großen Einfluss auf die Qualität der interpolierten Oberfläche hat.

Ein weiterer Aspekt wäre, in Anlehnung an die Arbeit von Wimmer et al. [27], die Einbindung von Bildinformation in das Verfahren. Eine Möglichkeit wäre auch hier, im ersten Schritt eine schnelle initiale Oberfläche zu erzeugen, um diese dann im zweiten Schritt, wie es Wimmer et al. gemacht haben, über Einbeziehung von Bildinformationen mittels Level Sets zu verfeinern.

Literaturverzeichnis

[1] J. Bloomenthal. Implicit surfaces. *Encyclopedia of Computer Science and Technology*, 1, 2001.

[2] I. Braude, J. Marker, K. Museth, J. Nissanov, and D. Breen. Contour-based surface reconstruction using mpu implicit models. *Graphical Models*, 69:139–157, 2007.

[3] M. D. Buhmann. Radial basis functions. *Acta Numerica*, 9:1 – 38, 2000. Cambridge University Press.

[4] M. D. Buhmann. *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, 2003.

[5] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH, pages 67–76. ACM, 2001.

[6] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell. Smooth surface reconstruction from noisy range data. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 119–126, 2003.

[7] J. C. Carr, W. R. Fright, and R. K. Beatson. Surface interpolation with radial basis functions for medical imaging. *IEEE Transactions on Medical Imaging*, 16:96–107, 1997.

[8] DKFZ Division of Medical and Biological Informatics. Geometry overview. URL: `httpdocs.mitk.org/nightly-qt4/GeometryOverviewPage.html`, [Stand: 13. Juni 2011]. [Last checked: 14. Juni 2011].

[9] DKFZ Division of Medical and Biological Informatics. Technical design of qmitksegmentation. URL: `httpdocs.mitk.org/nightly-qt4/QmitkSegmentationTechnicalPage.html`, [Stand: 13. Juni 2011]. [Last checked: 14. Juni 2011].

[10] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10:112–122, 1973.

[11] K. Ebisch. A correction to the douglas-peucker line generalization algorithm. *Computers and Geosciences*, 28:995–997, October 2002.

[12] FarField Technology Limited. Implicit surface modelling with fastrbf. URL: `http://www.farfieldtechnology.com/products/toolbox/theory/surfacefaq.html`, [Stand: August 2002]. [Last checked: 12. Juni 2011].

[13] H. Handels. *Medizinische Bildverarbeitung*. Vieweg+Teubner, 2 edition, 2009.

[14] T. Heimann and H.-P. Meinzer. Statistical shape models for 3d medical image segmentation: A review. *Medical Image Analysis*, 13:543–563, 2009.

[15] T. Heimann, B. van Ginneken, M. A. Styner, et al. Comparison and evaluation of methods for liver segmentation from ct datasets. *IEEE Transactions on Medical Imaging*, 28(8):1251–1265, Feb. 2009.

[16] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-15-7, http://www.itk.org/ItkSoftwareGuide.pdf, second edition, 2005.

[17] T. Lee and C. Lin. Feature-guided shape-based image interpolation. *IEEE Transactions on Medical Imaging*, 21(12):1479–1489, December 2002.

[18] B. S. Morse, T. S. Yoo, D. C. Chen, P. Rheingans, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Shape Modeling International*, pages 89–98, 2001.

[19] Y. Mrabet. File:human anatomy planes.svg. URL: `http://en.wikipedia.org/wiki/File:Human_anatomy_planes.svg`, [Stand: 7. Juni 2008]. [Last checked: 12. Juni 2011].

[20] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Transactions on Graphics*, 22:463–470, 2003.

[21] V. Savchenko, E. A. Pasko, O. G. Okunev, and T. L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14:181–188, 1995.

[22] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit An Object Oriented Approach To 3D Graphics*. Kitware Inc., 4 edition, 2006.

[23] G. Turk and J. F. O'Brien. Shape transformation using variational implicit functions. *SIGGRAPH*, pages 335 – 342, 1999.

[24] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396, Dec. 1995.

[25] H. Wendland. Computational aspects of radial basis function approximation. In K. J. et al. (eds.), editor, *Topics in Multivariate Approximation and Interpolation*, pages 231 – 256, 2005.

[26] Wikipedia. Gaußsches eliminationsverfahren. URL: `http://de.wikipedia.org/wiki/Gausssches_Eliminationsverfahren#Rechenaufwand_und_Speicherplatzbedarf`, [Stand: 3. Juni 2011]. [Last checked: 12. Juni 2011].

[27] A. Wimmer, G. Soza, and H. Hornnegger. Two-staged semi-automatic organ segmentation framework using radial basis functions and level sets. *3D Segmentation in Clinic*, pages 179 – 188, 2007.

[28] I. Wolf, M. Nolden, T. Böttger, I. Wegner, M. Schöbinger, M. Hastenteufel, T. Heimann, H.-P. Meinzer, and M. Vetter. The mitk approach. In *8th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI) - Workshop on Open-Source Software*, Palm Springs, USA, 2005.

[29] I. Wolf, M. Vetter, P. Hassenpflug, and H.-P. Meinzer. Extension of 2d segmentation methods into 3d by means of coons-patch interpolation. In *Seventh Korea-Germany Joint Workshop on Advanced Medical Image Processing*, 2003.

[30] I. Wolf, M. Vetter, I. Wegner, T. Böttger, M. Nolden, M. Schöbinger, M. Hastenteufel, T. Kunert, and H.-P. Meinzer. The medical imaging interaction toolkit. *Medical Image Analysis*, 9(6):594–604, December 2005.

[31] I. Wolf, M. Vetter, I. Wegner, M. Nolden, T. Böttger, M. Hastenteufel, M. Schöbinger, T. Kunert, and H.-P. Meinzer. The medical imaging interaction toolkit (mitk) - a tool facilitating the creation of interactive software by extending vtk and itk. In R. L. Galloway, editor, *Proceedings of SPIE Medical Imaging: Visualization, Image-Guided Procedures, and Display*, volume 5367, pages 16–27, San Diego, CA, USA, 2004.

[32] X. Wu, M. Y. Wang, and J. Chen. Narrow-band based radial basis functions implicit surface reconstruction. In *Geometric Modeling and Processing*, pages 519–525, 2008.